Worst-Case Execution-Time-Aware Parallelization of Model-Based Avionics Applications

Simon Reder, Fabian Kempf, Harald Bucher, Jurgen Becker, Panayiotis Alefragis, Nikolaos S. Voros, Stefanos Skalistis, Steven Derrien, Isabelle Puaut, Oliver Oey, et al.

Worst-Case-Execution-Time-Aware Parallelization of Model-Based Avionics Applications

Simon Reder * and Fabian Kempf † and Harald Bucher ‡ and Jürgen Becker §
Karlsruhe Institute of Technology, Karlsruhe, Germany

Panayiotis Alefragis¶ and Nikolaos Voros‖
University of the Peloponnese, Patra, Greece

Stefanos Skalistis** and Steven Derrien†† and Isabelle Puaut‡‡
Université de Rennes I (UR1), Rennes, France

Oliver Oey §§ and Timo Stripf ¶¶
emmtrix Technologies GmbH, Karlsruhe, Germany

Christian Ferdinand***
AbsInt Angewandte Informatik GmbH, Saarbruecken, Germany

Clément David†††
ESI Group, Paris, France

Peer Ulbig‡‡‡ and David Mueller§§§ and Umut Durak ¶¶¶
German Aerospace Center (DLR), Braunschweig, Germany

Multi-core processing systems are the solution of choice to provide high embedded computing performance, but drawbacks in timing-predictability and programmability limit their adoption in safety-critical aerospace applications. This work presents a compiler tool-flow for automated parallelization of model-based real-time software, which addresses the shortcomings of multi-core architectures in real-time systems. The flow is demonstrated using a model-based Terrain Awareness and Warning Systems (TAWS) and an edge detection algorithm from the image-processing domain. Model-based applications are first transformed into real-time C code and from there into a well-predictable parallel C program. Tight bounds

*Research Scientist, ITIV, Engesserstr. 5 76131 Karlsruhe, Germany.
†Research Scientist, ITIV, Engesserstr. 5 76131 Karlsruhe, Germany.
‡Research Scientist, ITIV, Engesserstr. 5 76131 Karlsruhe, Germany.
§Professor, ITIV, Engesserstr. 5 76131 Karlsruhe, Germany.
¶Assistant Professor, Electrical & Computer Engineering Department, Megalou Alexandrou 1, Koukouli, 26334, Patra, Greece.
‖Associate Professor, Electrical & Computer Engineering Department, Megalou Alexandrou 1, Koukouli, 26334, Patra, Greece.
**Research Scientist, Université de Rennes 1/IRISA, avenue du Général Leclerc - 35042 Rennes CEDEX.
††Professor, Université de Rennes 1/IRISA, avenue du Général Leclerc - 35042 Rennes CEDEX.
‡‡Professor, Université de Rennes 1/IRISA, avenue du Général Leclerc - 35042 Rennes CEDEX.
§§Senior Engineer, emmtrix Technologies GmbH, Haid-und-Neu-Straße 7, 76131 Karlsruhe.
¶¶Managing Director Technology, emmtrix Technologies GmbH, Haid-und-Neu-Straße 7, 76131 Karlsruhe.
***CEO, Science Park 1, 66123 Saarbruecken.
†††Senior Software Developer, Scilab Enterprises, 143 bis rue Yves Le Coz, 78000 Versailles, France.
‡‡‡Research Scientist, Institute of Flight Systems, Lilienthalplatz 7, 38108 Braunschweig.
§§§Research Scientist, Institute of Flight Systems, Lilienthalplatz 7, 38108 Braunschweig.
¶¶¶Research Scientist, Institute of Flight Systems, Lilienthalplatz 7, 38108 Braunschweig, and Senior Member.

for the Worst-Case Execution Time (WCET) of the parallelized program can be determined using an integrated multi-core WCET analysis. Thanks to the use of an architecture description language, the general approach is applicable to a wider range of target platforms. An experimental evaluation for a research architecture with network-on-chip (NoC) interconnect shows that the parallel WCET of the TAWS application can be improved by factor 1.77 using the presented compiler tools.

Nomenclature

NS_i = A sequence of consecutive basic blocks

ϵ_i = Worst-case end time for node sequence NS_i in clock cycles

σ_i = Worst-case start time for node sequence NS_i in clock cycles

I. Introduction

The process of digitalization leads to an increasing adoption of information and communication technologies in various areas, including the aeronautics domain. This development is visible in emerging technologies such as connected flight, autonomous air vehicles or intelligent drone-swarms. Literature like [1] even predicts that smart technologies will lead to a revolution of the entire aeronautics area, the so-called *Flight 4.0*.

A fundamental precondition for these innovations is the availability of sufficient computational performance. The solution of choice to meet this requirement are heterogeneous multi- and many-core hardware platforms that offer a good trade-off between performance and energy efficiency. The downside, however, is that the safety requirements for electronic systems in the aeronautics domain are much harder to satisfy when multiple processor cores operate in parallel. One major problem is the verification of hard real-time guarantees for parallel software that runs on a multi-core system. Time-critical hardware/software systems must be proven to react within a given time frame regardless of its current state or input data. This property has to be verified in the design phase of the system, where a safe upper bound for the worst-case execution time (WCET) must be determined. Safe WCET bounds, however, can only be guaranteed by means of a static analysis of the program binary with respect to the hardware platform. Profiling-based approaches are not sufficient, as it is usually impossible to prove that the assessed execution scenario matches the worst possible case. While static WCET analyzers based on Abstract Interpretation (e.g. [2]) are commercially available for a range of single-core processors, state-of-the-art solutions for multi-core systems only exist for a few specialized hardware platforms.

The lack of multi-core WCET-analysis tools results from open challenges in predicting the temporal behavior of multiple tightly-coupled processor cores. The main problem are concurrent accesses to shared hardware resources from different cores [3]. In presence of such accesses, a piece of code running on one of the processors can affect the

execution times of other cores. If the processors are e.g. connected to a shared memory bus, the arbitration unit might delay some accesses while the bus is blocked by other cores. This so-called *timing interference* is hard to predict at compile-time and pessimistic assumptions like the permanent presence of conflicting accesses might be required to get safe WCET bounds. Such assumptions, however, can easily eliminate the benefit of parallelism for the worst-case execution scenario, since shared resources become a limiting bottleneck.

Previous works address the interference problem either using specialized multi-core hardware that is fully *timing compositional* or through approaches to predict and bound interference costs at design time. In the former case, the hardware ensures that concurrent accesses from different cores cannot affect each other's access times [3]. This is usually achieved by ensuring a fixed spatial or temporal separation of concurrent accesses. Spatial separation can, for instance, be realized with separated memory busses per core, while temporal isolation is e.g. provided by static *Time-Division Multiple Access (TDMA)* arbitration schemes. In both cases, timing-compositional multi-core hardware suffers from drawbacks such as the increased hardware costs for multiple memory busses or the performance bottlenecks due to bandwidth sharing in TDMA.

As an alternative to timing compositional hardware, the prediction of interference costs at design time has been studied in several previous works like [4–7]. This approach puts lower burdens on the hardware platform but requires sophisticated static WCET analyzers, restricted programming models and extensive program optimization towards reduced interference costs. To eliminate these drawbacks, interference analysis can be combined with compiler-based program-optimization and automated parallelization. This way, automated compiler tools can handle the necessary parallelization, optimization and adaption steps, while the WCET-analysis phase may re-use information from the compilation stage to improve its accuracy. While the tight integration of WCET-analysis and automated parallelization has not been explored in detail by previous works, this article presents a novel solution in the form of the integrated "ARGO" tool flow*. The flow takes high-level programs, model-based applications or C code as input and produces a functionally equivalent parallel C program. It combines model-based design, WCET-aware automated parallelization, and interference analysis to automate the necessary optimization of parallel programs towards low interference and high predictability. In this way, the ARGO tools simplify the development and verification of parallel real-time software while allowing developers to take advantage of model-based design principles.

Various avionics systems used in modern aircraft can benefit from automated parallelization solutions. In this work, we demonstrate the applicability of the proposed tools and evaluate its promises by means of a Terrain Awareness and Warning System (TAWS), which is based on the example of the Enhanced Ground Proximity Warning System (EGPWS)†. A TAWS provides visual and aural warnings, which aim to prevent Controlled Flight Into Terrain (CFIT) type of accidents. EGPWS is one of the various TAWS options available in the market. It has five basic modes

*`http://www.argo-project.eu`

†EGPWS is a TAWS from Honeywell: `https://aerospace.honeywell.com/en/pages/enhanced-ground-proximity-warning-system`

that lead to visual and aural warnings between 30 ft (9.144 m) to 2450 ft (746.76 m) Above Ground Level (AGL). Examples would be warnings for excessive descent rates for all phases of flight (Mode 1), or significant altitude loss after take-off or during a low altitude go around (Mode 3). Additionally, an EGPWS provides some enhanced functions like the "Terrain Alerting and Display" and the "Terrain Look Ahead Alerting", which rely on a terrain elevation database.

As the main contribution of this article, we present the first complete prototype of the ARGO tool-flow together with the model-based TAWS as a real-world use-case application. In contrast to previously published concept work [8, 9], we demonstrate the feasibility of model-based design principles for time-critical avionic applications like TAWS and show that parallelization can help to reduce WCET-bounds. To demonstrate the interference prediction capabilities of the presented tools, our experiments with the TAWS are complemented by an evaluation for a computationally more intensive image-processing algorithm. We measure the benefit of multi-core platforms for real-time applications, by comparing the WCET of the parallelized C code with the WCET of an equivalent sequential implementation that is generated before parallelization. The experiments are carried out for a multi-core research architecture consisting of time-predictable Leon3 processor cores and a network-on-chip (NoC) interconnect.

The remainder of this article is structured as follows: Section II presents related work in the area of model-based design in avionics, tool-assisted parallelization of embedded software and multi-core WCET analysis. The model-based TAWS application is introduced in Section III. Section IV presents the proposed parallelization approach, followed by a description of the multi-core WCET analysis tool in Section V. Section VI presents our experimental results before Section VII concludes the article.

II. Related Work

Considerable previous works studied parallel software for flight systems with emphasis on the applicability of multi-core architectures with respect to the safety constraints of the avionics domain [10–12]. Definitely, segregation, integrity, predictability, certification costs and performance are important challenges to tackle [13]. Notwithstanding, an effective and efficient development methodology for avionics applications using multi-core architectures still remains as a research question. The aerospace domain is yet to adopt complex toolchains and programming processes for exploiting the full potential of these next-generation heterogeneous parallel platforms.

Automated and tool-assisted software parallelization in different application domains has been an active research field for the last decades. Existing solutions in the field of High-Performance Computing (HPC) often target standard APIs like *OpenMP*[‡] or the *Message Passing Interface* (MPI)[§], which are not designed to meet hard real-time requirements.

[‡] `https://www.openmp.org/`
[§] `https://www.mpi-forum.org/`

Therefore, we will focus in this section, on the related works [14–18], which are dedicated to the domain of embedded cyber-physical systems.

The MAPS tool presented in [14] transforms the input program into a parallel intermediate representation that is based on a *Process Network* (PN) model of computation. This representation is used to apply various program transformations before an optimized parallel C code is produced as output. Different heuristics are used in the process to identify parallelism and profiling information serves as (average-case) performance measure.

The approaches [15] and [16] as well as the ALMA tool-flow in [17] use the *Hierarchical Task Graph* (HTG) proposed by Girkar et al. in [19] to extract task-level parallelism from the input program. The HTG representation is then used to generate a parallel schedule that is optimized to improve the average-case execution time.

The approach of the parMERASA project presented in [18] aims for WCET-aware parallelization based on parallel algorithm design patterns. It includes tool-support for application developers to fit their algorithms into parallel and predictable design patterns and algorithm skeletons. From these patterns, real-time guarantees are derived for the parallel execution of the algorithms on a multi-core platform.

Although some of the previous works successfully applied the HTG model for automated parallelization with average-case measures, the presented flow is the first to use it for WCET-aware parallelization. Except for the pattern-based approach in [18], the existing works aim to improve the average-case performance rather than the WCET. This limits their usability for hard real-time systems, where the WCET is the deciding criterion. Different from the average-case approaches, the ARGO flow is designed to address the predictability and interference challenges that are immanent to hard real-time systems. This is essential to enable tight WCET-analysis for hardware that is prone to timing interference.

Similar to ARGO, the parMERASA approach [18] explicitly targets hard real-time systems. However, the semi-automatic pattern-based approach is limited to a relatively coarse granularity, while the ARGO tools provide fine-grained parallelization with a higher degree of automation. Instead of relying on the end-user to apply pre-defined parallel design patterns, ARGO generates parallel programs in a largely automated process, while user decisions can still be included to improve the results.

The mentioned previous parallelization tools are all based on the C programming language and only [17] allows for a higher-level programming language as input. In contrast to that, the presented tool flow accepts model-based applications and high-level Scilab code as additional input languages. This reduces the expertise an end-user needs in order to implement, test and parallelize hard real-time software. Due to the predictability requirements, the ARGO tools, unlike approaches for average-case parallelization, need to impose additional restrictions on the input programs. The flow requires a well-predictable subset of C99, which enforces e.g. the absence of dynamic memory allocation and pointer arithmetic. Such limitations maintain the static WCET analyzability and are already common in real-time software for single cores. The possibility to use high-level programming languages helps to hide these restrictions from

the end-users.

Related research on WCET estimation of multi-threaded code is presented in [20, 21]. In [20], the analyzed software consists of manually parallelized code, which is annotated such that the WCET estimation tool can identify task and inter-task synchronization (barriers, critical sections). The ARGO tool-flow, in contrast, integrates the automated task extraction process with the system-level WCET calculation and thus eliminates the need for error-prone manual annotations. The research reported in [21] proposes to integrate the calculation of synchronization costs using a core-level WCET estimation tool, which allows to capture hardware effects across tasks assigned to the same core. In contrast to [21], the technique designed for ARGO supports a more general execution model (some communications inside loops were difficult to deal with in [21]) and additionally copes with contention when accessing shared resources.

A lot of research has been performed on dealing with the cost of contention when accessing shared resources in real-time systems, both on applications modeled as independent tasks [22–24] and applications modeled as task graphs [4–7]. In comparison with related work, the ARGO approach identifies code sections that interfere with one another on non-trivial task graphs (including loops). Moreover, the ARGO approach is platform agnostic thanks to the use of an Architecture Description Language (ADL).

To the best of our knowledge, the presented approach is the first to integrate model-based design principles, a fine-grained automated parallelization for hard real-time systems and a static multi-core WCET-analysis into a holistic tool-flow. While parts of the tool-flow rely on previously published concepts [8, 25, 26], this article is the first to present results for the integrated approach including the multi-core WCET analysis.

III. Model-based Terrain Awareness and Warning System

The model-based design is characterized by the use of executable graphical data-flow oriented block diagram models and state machines for system specification, design, and implementation [27]. It promotes mathematical modeling to design, analyze, implement and evaluate dynamic systems and endorses extensive use of simulation in model-in-the-loop, software-in-the-loop, processor-in-the-loop, hardware-in-the-loop, and, in summary, x-in-the-loop fashion for verification purposes. The ARGO model-based design workflow starts with developing application models in Scilab/Xcos. Scilab/Xcos[¶] is an open-source model-based design and simulation environment [28]. Other model-based design and simulation environments include ANSYS SCADE Suite[‖] and MATLAB/Simulink[**], both of which are commercial and broadly used for avionics applications.

Following the ARGO design flow, the basic modes of the TAWS, which correspond to the basic modes of the EGPWS, have been modeled using Scilab/Xcos. The algorithms for the enhanced functions such as Terrain Alerting

[¶] https://www.scilab.org/software/xcos
[‖] https://www.ansys.com/products/embedded-software/ansys-scade-suite
[**] https://www.mathworks.com/products/simulink.html

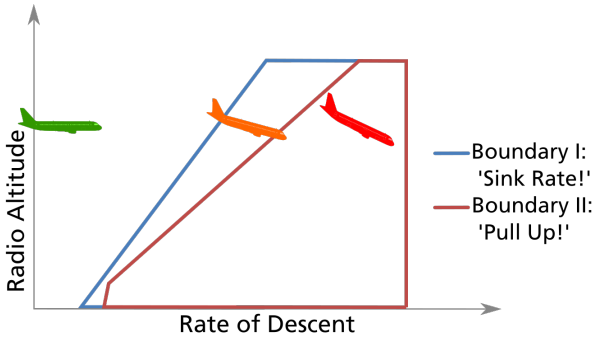


Fig. 1 Mode 1: Excessive Descent Rate.

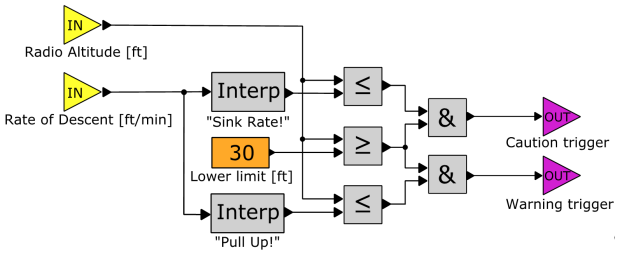


Fig. 2 Simplified Scilab/Xcos diagram for Mode 1.

and Display and Terrain Look Ahead Alerting are implemented within the model as Scilab scripts.

Figure 1 presents the Excessive Descent Rate mode as an example. As demonstrated by their position in the graph and their orientation, the three aircraft have the same altitude, but different rates of descent. While the green aircraft is in a safe flight state, the orange one's rate of descent causes a warning. The red aircraft's descent rate, however, is too high considering its low altitude, requiring immediate action by the pilot. Figure 2 shows an overview of the implementation of this mode in Scilab/Xcos.

Figure 3 depicts the top-level structure for the entire TAWS model. Using 22 different Scilab/Xcos block types, it has in total over 800 blocks and 40 superblocks. In this representative avionics application use case, Scilab/Xcos provided an adequate set of features for model development [29]. Code generation and parallelization features of the ARGO workflow furnished a positive user experience. However, the infrastructure for industry-grade simulation-based verification and model quality assurance required an extra effort. The details of simulation-based verification approaches for x-in-the-loop testing can be found in [30, 31]. The utilization of flight simulators for verification of model-based avionics applications on multi-core targets is further discussed in [32]. The concerns regarding the quality assessment of the models are further discussed in [33].

IV. WCET-aware Parallelization Tool-Flow

Figure 4 depicts the basic structure of the ARGO tool-flow. The parallelization part consists of the three steps *Code Transformation*, *Scheduling & Mapping* and *Data Management & Synchronization*. While a similar decomposition of the parallelization problem has been proposed in existing solutions like [17], novel and vastly different algorithms are required to parallelize hard real-time applications efficiently. The tight integration of the static System-Level WCET (SL-WCET) analyzer with the compiler flow furthermore advances the state of the art in multi-core WCET-analysis tools. Embedded into a holistic cross-layer optimization loop (left side of Figure 4), the components can iteratively exchange feedback information and enable user-interactive solution space exploration.

As described in the previous sections, the input programs for the ARGO tool-flow can be given as a combination of

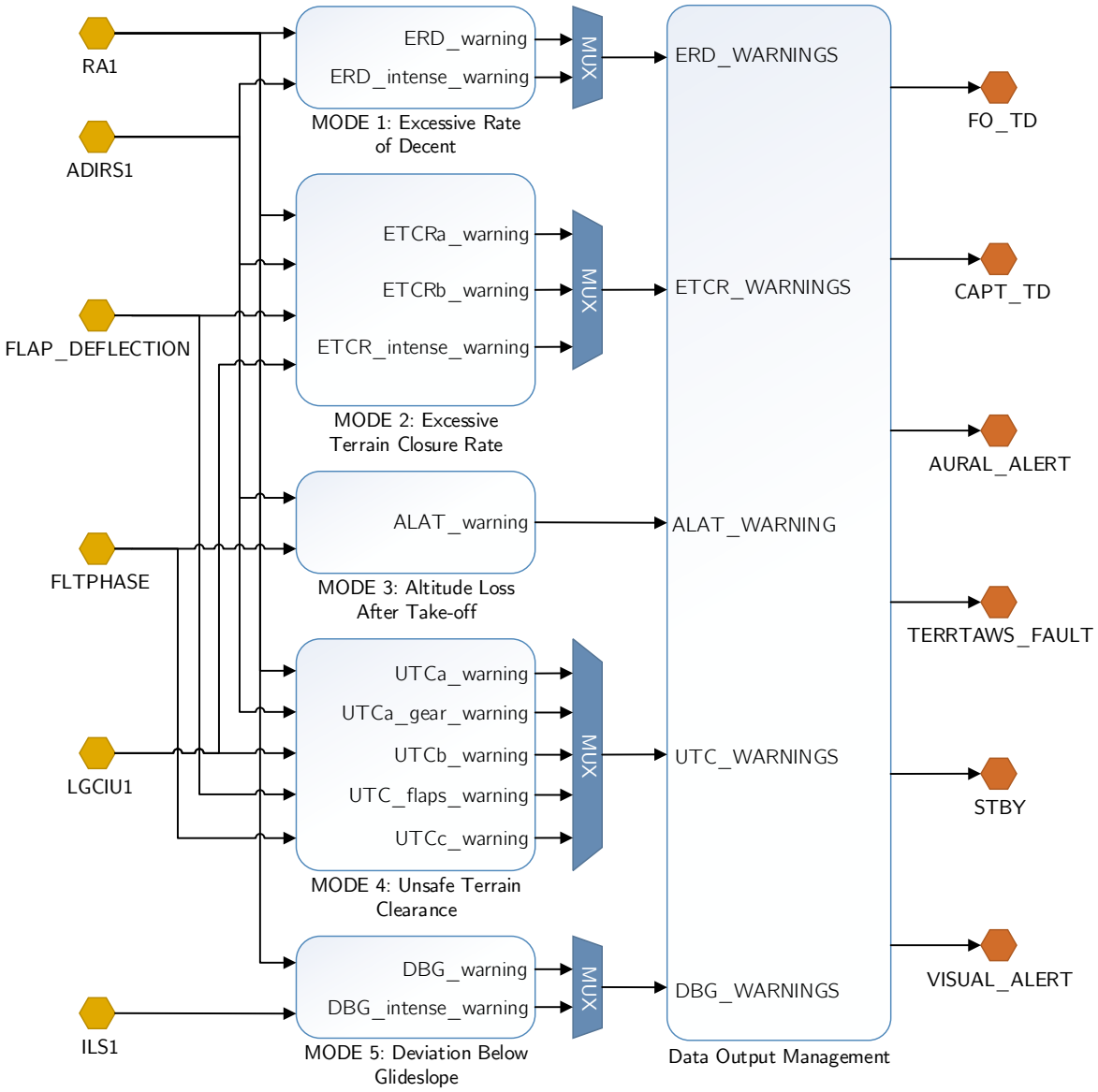


Fig. 3 Top-level structure of the TAWS Xcos model.

Xcos, *Scilab* and predictable sequential C artifacts. The *Scilab/Xcos Front-End* initially transforms all Scilab/Xcos parts into a generic platform-independent C implementation. The (generated) C application and its corresponding ARGO intermediate representation is the starting point for the parallelizing source-to-source compiler flow. The purpose of the subsequent code-transformation step is to expose parallelism from loop nests and to improve the data locality. The latter is crucial to minimize potential sources of timing interference by reducing the need for communication and memory resource sharing. The program intermediate representation after code transformations is still sequential but contains a higher number of independent program parts for efficient parallelization.

The scheduling and mapping component solves the optimization problem of mapping reasonable parts of the program to the available cores and generating a static schedule that aims to minimize timing interference. The step uses

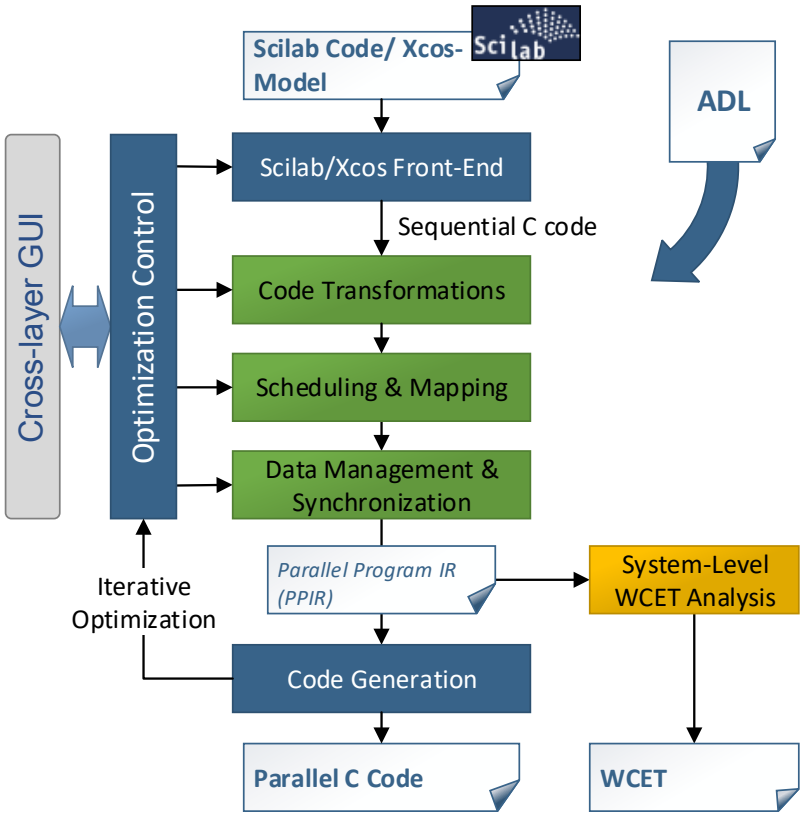


Fig. 4 Structure of the ARGO tool-flow.

the well-known *Hierarchical Task Graph* (HTG) representation [19] as an abstraction layer to model the optimization problem. The data management & synchronization step finally transforms the sequential program into a parallel one, which efficiently implements the generated HTG schedule. This step does not rely on the HTG abstraction but uses control flow graphs (CFG) for fine-grained optimization of the memory map and the synchronization structure. The output is a *Parallel Program Intermediate Representation* (PPIR), which follows a WCET-aware programming model with FIFO-based communication [25]. Depending on the hardware platform, the PPIR may use shared memories in addition to message-based FIFO-communication to exchange data between cores.

For the generated parallel program, the SL-WCET analyzer finally computes a safe upper bound for the multi-core WCET. It uses knowledge preserved from the parallelization stages (especially the synchronization structure) in order to bound the number of conflicting accesses that may possibly occur at the same time. Besides the safe overall WCET, more detailed information about worst-case memory accesses and approximated WCET contributions of individual program parts is returned as feedback information for later iterations of the parallelization flow.

Static WCET analyzers like the ARGO SL-WCET module require knowledge of all timing-relevant hardware details to produce reasonable results. Besides the microarchitecture of the cores, the temporal behavior of interconnects is of particular interest in the multi-core case. To cover the variety of different bus and network-on-chip (NoC) topologies in modern multi- and many-core architectures, the ARGO flow relies on a flexible platform architecture model. This model

is embedded into a customized Architecture Description Language (ADL) format based on the *Software-Hardware Interface for Multi-Many-Core 1.0* (SHIM) specification[††]. The original SHIM specification has been extended for the proposed tool-flow to enable detailed modeling of communication components and interconnects. The extended model especially contains a description of routing paths, arbitration schemes, cache behavior, data throughput, and latencies to provide the necessary information for interference cost computation. The ADL is used in the ARGO flow by the parallelization steps to guide the optimization algorithms and by the SL-WCET to extract worst-case timings for memory accesses and interference effects.

The described steps (or layers) of the basic ARGO flow each expose a range of decisions, input parameters, and solver algorithms. These inputs are supplied by the so-called *Cross-Layer Optimization Control* components shown on the left side of Figure 4. The goal is to improve the overall parallelization result by co-optimizing the input decisions across all steps/layers in a user-interactive process. We use an iterative approach to account for the fact that decisions in earlier steps may heavily affect available decisions and degrees of freedom in later steps. Code transformation decisions, for instance, might substantially change the program structure including the task graphs used by the scheduler. The impact of these effects on the final SL-WCET is hard to model when making decisions for one step in isolation. For that reason, the cross-layer system back-propagates SL-WCET results from previous iterations, which can be used to refine decisions and to eliminate bottlenecks. The end-user can interactively guided this process using an integrated graphical user interface (GUI).

A. Code Transformations

The goal of the code-transformation step is to expose the inherent parallelism of nested loop structures and to improve the data locality. This is especially useful for algorithms that operate on large arrays that can be split into independent slices. Such slicing transformations can increase the number of independent tasks in the HTG and thus allow the scheduler to parallelize the transformed slices-processing routines. Another advantage is that smaller array slices are more likely to fit into fast core-private scratchpad memories, which can help to avoid interference costs for shared-memory accesses.

The transformations are realized using polyhedral compilation techniques, which are able to optimize larger code fragments with several different loop nest structures. The effect of these polyhedral transformations on the WCET bounds of sequential programs has already been studied with promising results in [26]. However, the TAWS application considered in this article is control-flow-dominated, without extensive computations inside loops. Thus, polyhedral transformations have no significant impact on this use-case, which is why their effect on parallelized programs is not studied in detail within this work. For more information about the code transformation approach, we refer to [26].

[††]SHIM is standardized by the Multicore Association: `http://www.multicore-association.org/workgroup/shim.php`

B. Scheduling & Mapping

The scheduling and mapping step solves the optimization problem of deciding which parts of the input code will run in parallel on which processor core. The Hierarchical Task Graph (HTG) [19] serves as an intermediate representation to formulate the optimization problem in a well-defined way.

1. Hierarchical Task Graph Model

```
int htg(int a[256], int b[256], int x[128], int y[128][128]) {
  int c, i, j, sum;
  c = a[255] * b[255];
  for (i=0;i<128;i++) {
    x[i] = a[i] * b[i];
    for (j = 0; j < 128; j++)
      y[i][j] = a[i]*b[j];

  }
  sum = 0;
  for (i=1;i<128;i++)
    sum += a[i];
  return sum / c;
}
```

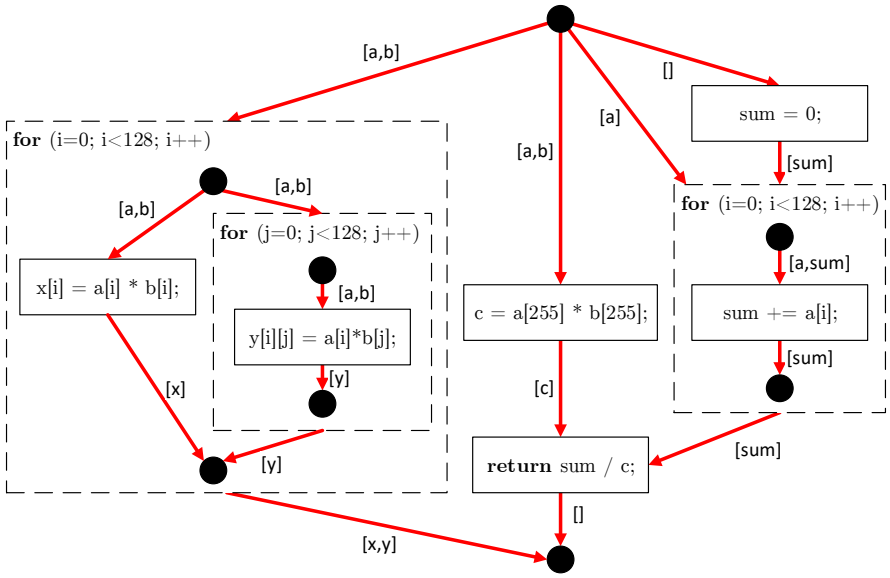


Fig. 5 A source code example and the corresponding HTG with data dependencies for the variables

The HTG is an abstraction of the sequential program representation that is suitable to generate optimized parallel schedules. The main advantage of the HTG is that its hierarchical structure resolves cyclic data dependencies, which are common in programs with loops. The absence of cycles, in turn, is an important prerequisite for many scheduling algorithms. The HTG model clusters dependency cycles (i.e. strongly connected components) into separate hierarchy levels, such that each hierarchy level represents a directed acyclic graph (DAG). Figure 5 illustrates that for an exemplary code fragment and the corresponding HTG. The hierarchy levels have distinct start and end nodes (circles in the figure),

while the HTG edges represent data dependencies. Given the hierarchical structure, the HTG can encapsulate parallelism on all levels of granularity, including the innermost loop nests. For the depicted example, the HTG immediately reveals, that there can be at most four task nodes running in parallel at the same time. In the absence of recursion, the HTG can be extended to cover multiple functions using virtual inlining of function calls.

Schedules for the HTG can be generated by recursively solving the individual acyclic hierarchy levels with bottom-up or top-down strategies. In doing so, a scheduler can explore and potentially parallelize the entire hierarchical structure without the need to handle cyclic dependency graphs.

2. Mapping & Scheduling of HTG Tasks

The goal of task mapping and scheduling is to identify potential parallelization opportunities, estimate the interference caused by tasks running in parallel and subsequently decide on the optimal allocation and sequencing of tasks to processing cores. Data mapping is the decision making process that places variables required by executing tasks to the available memory regions (e.g. scratchpad, shared memory or main memory) and affects the execution time and the interference between tasks. The aforementioned decisions are based on the estimated WCET of each task, which is gained by a sequential WCET-analysis pass. Unfortunately, the WCET of the tasks may change with the current schedule, which creates a cyclic dependency between WCET-estimation and scheduling decisions. One approach is to solve the problem in a single optimization model, which is not scaling to larger problem instances but can provide an optimal solution, given enough time.

The ARGO approach is to provide multiple algorithms with different performance/quality characteristics that share a common input/output format, thus creating a toolset of interchangeable algorithms. The implemented algorithms range from list-based heuristics (HEFT-LA, CPOP)[34, 35] for very fast good-quality solutions to meta-heuristics with better solution quality and finally to ILP-based solvers that generate optimal or near-optimal solutions [36], but may take a very long time for bigger problem instances. As they share a common problem representation, all algorithms can be applied to HTG subtrees or individual DAGs to provide alternative solutions or to be used as lower-level algorithms in a hyper-heuristic decision process. The major changes of the heuristic algorithms compared to the literature is that the execution times of all tasks, the current critical path as well as the priority list of remaining tasks are updated with each task insertion in the solution. When deciding the mapping of a top task in the priority list, a multi knapsack problem is solved for each core to calculate a mapping of the variables used by the task to the accessible memory areas. This is due to the fact that, depending on the variable mapping, every individual task insertion may create interference with all other tasks that may execute in parallel on other cores. For the TAWS use case, a 20% reduction in the WCET estimation was observed when calculating memory mapping before each task placement. In our approach, variable mapping remains constant for the complete execution of each task. The detailed ILP model for the scheduling problem is presented by Alefragis et al. [9].

User-defined constraints on specific mapping of tasks to processing cores, maximum parallelization per task, clustering of tasks and variable mappings to specific memory segments are supported by all algorithms. For example, the end-user may define a group of tasks that should be assigned as a continuous sequence to a single processor core to prevent them from interfering with each other. Different solution strategies may also be selected for specific parts of an HTG to combine multiple algorithms. The scheduling algorithms may furthermore include SL-WCET information as feedback from previous iterations of the cross-layer optimization loop (see Subsection IV.D). At the beginning of each iteration, the feedback information is compared to the estimated execution time of tasks in the previous solution, as well as the interference between tasks that may execute in parallel and the starting/ending time of communication operations. When major differences are observed, the execution-time estimation is updated to reflect the difference and the tasks may be tagged not to be scheduled in parallel to different cores. The updated information is subsequently used to generate an alternative solution in the next iteration. In addition, end-users may also influence the behavior of algorithms with more advanced constraints. For example, when high interference between potentially parallel tasks is detected, the user may insert a "scheduling" constraint to prevent them from being scheduled in parallel.

C. Data Management & Synchronization

The *Data Management & Synchronization* step transforms the sequential program into a parallel one that implements the given schedule. The step can be broken down into the four phases 1) *Data Partitioning*, 2) *Synchronization Refinement*, 3) *Deserialization* and 4) *Memory Mapping*. To guarantee the absence of deadlocks by construction, most transformations are applied to the sequential Control Flow Graph (CFG) before deserializing it into separate CFGs per core in phase 3).

The goal of phase 1) is to subdivide the data fields of the original program into separate sets of private variables for each processor core. This is done with respect to the mapping decisions made for the tasks of the HTG model. Partitioning the data fields enables the use of distributed memory paradigms, where variables are stored in core-private memories with explicit message-based communication for data exchange. New local variables are created if necessary and communication operations are inserted in order to preserve the original data-flow. While distributed memory is the default case, an optimization algorithm may decide to use shared memory semantics for selected variables in order to avoid extensive communication overhead. The last step of phase 1) reorders the sequential control-flow to match the order specified by the scheduled HTG.

Phase 2) optimizes the placement of communication and synchronization operations. The HTG model assumes communication to take place at the beginning and/or the end of task nodes as shown in the upper part of Figure 6. The goal of phase 2) is now to analyze data dependencies on a finer level of granularity in order to move communication operations to more suitable positions without being limited to task boundaries. The lower part of Figure 6 illustrates how this can help to close gaps in the schedule. The optimization pass considers three different criteria to select the new

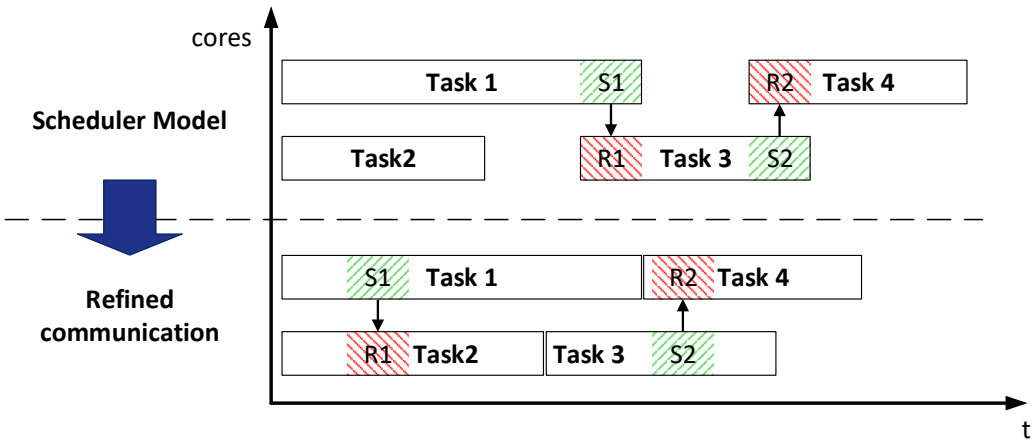


Fig. 6 Example schedule with send (S) and receive (R) operations before and after Synchronization Refinement

positions: I) the position should be suitable to close gaps in the schedule, II) the worst-case execution count for the chosen position should be as low as possible (e.g. outside of loop nests), and III) redundant synchronization operations should possibly be moved to the same position in order to merge them into a single one. Redundant synchronization can occur in the scheduled HTG e.g. if two independent pairs of tasks require synchronization between the same pair of cores. When moving communication points, both the sending and the receiving operation are always re-positioned together. In doing so, the optimization step cannot introduce backward dependencies that would later result in deadlocks.

Phase 3) deserializes the sequential control flow into one separate CFG per core. This is done by replicating the control flow structure for each core while leaving out nodes (i.e. basic blocks) that are mapped to other cores. To account for control-flow dependencies, synchronization is inserted for all branches and loops that affect send/receive operations. In doing so, we ensure that a send and the corresponding receive operation are always executed with the same execution count.

For the resulting parallel program, phase 4) finally computes a mapping of the program variables to the memories of the target platform. It aims for minimal interference on shared memories, busses and interconnects. The memory mapper relies on the results of a may-happen-in-parallel (MHP) analysis to predict the interference costs and uses ILP to model and solve the optimization problem.

The *Parallel Program Intermediate Representation (PPIR)* generated by the Data Management & Synchronization steps can directly be transformed into parallel C code. The PPIR together with a compiled binary of the generated code is used as input for the multi-core WCET analysis.

D. User-Interactive Cross-Layer Optimization

The iterative cross-layer process of the ARGO flow enables efficient decision-space exploration across all tool-flow steps (layers). A key element in this process is to maintain the back-traceability of basic blocks between the PPIR, HTG

and CFG representations. This is achieved by means of persistent identifiers for basic blocks, which are preserved by the transformation steps. These identifiers enable a traceability mechanism that allows information on basic-block level to be exchanged between earlier and later intermediate representations. We use that, e.g., to back-annotate local WCET information from the final SL-WCET to the earlier intermediate representations in subsequent tool-flow iterations. Due to interference, cache effects and the generated memory map, this feedback information can significantly differ from the *a priori* WCET, which is derived from the sequential program. Therefore, knowing both the *a priori* and the feedback information can help to improve the solutions generated by the scheduling and the communication optimization steps.

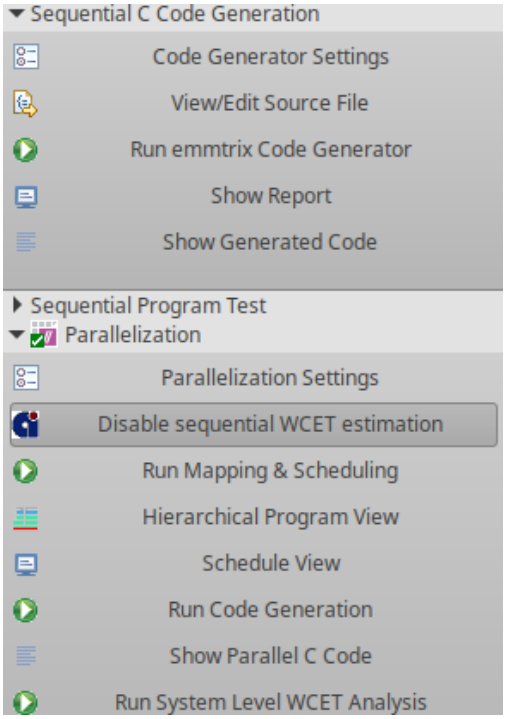


Fig. 7 GUI of ARGO workflow

The traceability features and the SL-WCET feedback information are also useful for the user-interactive decision-space exploration process. This process is controlled through a graphical user interface (GUI) that visualizes the results of the different steps. The GUI can be used to tweak decisions like the mapping of HTG tasks or the application of specific loop transformations iteratively. The ARGO tool-flow is mapped to a view in the GUI as shown in Figure 7. During the "Sequential C Code Generation" step, the input Xcos model and Scilab code is converted to C code. The "Parallelization" step contains the sequential WCET estimation, the automatic call to the WCET-aware scheduler, a hierarchical program view, a view of the schedule, the automatic parallel code generation and the system-level WCET. All passes are executed automatically while taking into account all settings and user constraints defined either as pragmas in the C code or by setting them in the GUI. The hierarchical program view is another central view, which visualizes the hierarchical task graph described in Section IV.B.1. It allows the end-user to interact with the different layers of the tool

flow:

- *Code transformations*: Where applicable, users can select code transformations that should be applied to the selected block. Depending on the transformation, these changes are applied during the sequential C code generation or directly on the C code. They include simple transformations like loop fission or splitting, as well as complex polyhedral transformations.
- *Mapping constraints*: Users can apply mapping constraints by specifying on which core a task should be executed, excluding cores from the automatic scheduling/mapping algorithms or by adding cluster constraints that limit the parallelization of tasks.
- *Scheduling results*: The results of the scheduler are displayed in the schedule view as shown in Figure 8. It shows the load of the individual cores of the target platform across time.
- *Data management*: Data dependencies can be analyzed using the schedule view, as shown by the arrows in Figure 8.

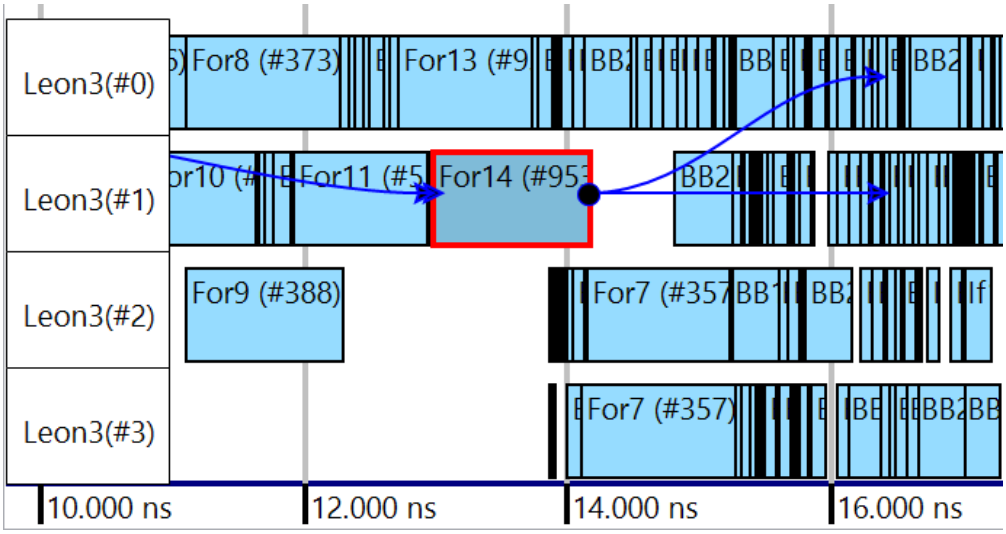


Fig. 8 Schedule view

V. Multi-core WCET-analysis

In general, static WCET estimation tools operate on sequential code snippets, that are meant to be executed on a single core of the architecture, and thus do not attribute any blocking delays or any interference delays caused by activities on the other cores. Our approach to multi-core WCET analysis is to use such a state-of-the-art sequential WCET analysis on core-level and enhance it with the aforementioned delays to get a System-Level WCET (SL-WCET). On core-level, we utilize the *aiT* static WCET analyzer [37], which has been used in the avionics industry for more than a decade. The tool uses the technique of *abstract interpretation* to directly analyze binary executables taking into account the core's micro-architecture (cache and pipeline). The analyzer is used in three steps of the ARGO workflow:

(i) to estimate the Core-Level WCET of code snippets, (ii) to obtain the worst-case number of memory accesses and (iii) to calculate a System-Level WCETs (after adding the delays caused by the parallel execution of program parts on other cores).

A. System-Level WCET Analysis

Since, by their construction, static WCET analysis tools calculate WCET estimates for code running on a single-core, out-of-core induced delays have to be computed separately. For a parallel application as generated by the ARGO toolchain, two types of delays have to be added to the sequential WCET estimation provided by the core-level analysis:

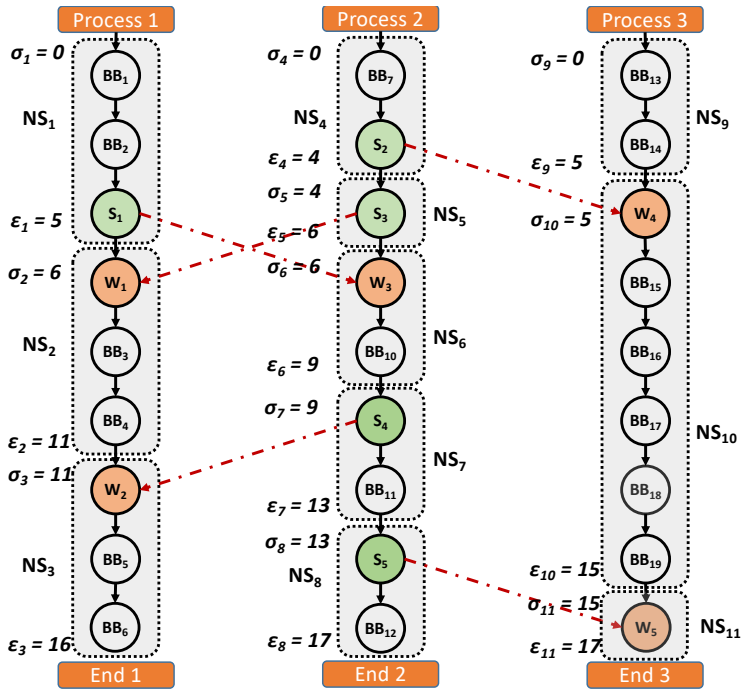
- Delays due to interference when accessing shared hardware resources such as buses and shared memories
- Communication/Synchronization delays, when the execution on one core waits for data from another core

Calculation of these delays is achieved on the Scheduled PPIR, an example of which is given in the left part of Figure 9 for a 3-core architecture. Circles represent basic blocks, plain arrows represent the control flow within each core and dashed arrows represent unidirectional communication between cores. Colored blocks labeled S and W represent Signal/Wait blocks and model the endpoints of communication/synchronization between the cores (Send/Receive operations can be seen as a special case of Signal/Wait). For simplicity, the selected running example does not include loops and branches, although they are supported in the PPIR format and the whole ARGO tool-chain. The role of shaded areas grouping blocks and notations σ and ϵ will be explained later.

In ARGO, accesses to the shared resources are arbitrated with particular policies (described in the ADL), and their time to complete may highly depend on the code snippets running on the other cores. To calculate delays caused by interference when accessing shared resources, the System-Level WCET performs several analyses in the following order: **Step 1. Acyclic-PPIR Graph Construction:** As a first step, to aid the May-Happen-in-Parallel analysis (see Step 3), the PPIR graph is transformed into an acyclic one. This is achieved with a graph transformation algorithm, which unrolls the loop body twice and removes the loop's back-edges.

Step 2. Basic-block Sequence Graph Construction: Following, consecutive basic blocks (on the same core) of the acyclic-PPIR graph are grouped into *sequences*, corresponding to code-snippets. Essentially, a sequence is the ordered set of basic blocks between two synchronization (Signal/Wait) nodes. Sequences are represented as grey areas grouping blocks in the left part of Figure 9.

Step 3. May Happen in Parallel (MHP) analysis: Subsequently the MHP analysis is performed to identify which of the (previously constructed) sequences may interfere with other sequences. It does so by constructing the transitive closure of the acyclic-PPIR graph, considering both intra-core and inter-core communications; two sequences NS_i and NS_j may happen in parallel if there is no edge between them in the transitive closure. In Figure 9, for example, NS_3 can potentially happen in parallel to NS_9, since there might be an execution scenario where NS_3 starts before the worst-case end time ϵ_9 of NS_9 (σ_3 is only an upper bound). Without knowing a safe lower bound for the start time of NS_3, only a



Seq.	MHP	Interference	WCET
NS_1	$NS_{\{4,5,9,10,11\}}$	3	2
NS_2	$NS_{\{6,7,8,9,10,11\}}$	3	2
NS_3	$NS_{\{7,8,9,10,11\}}$	3	2
NS_4	$NS_{\{1,9,10\}}$	2	2
NS_5	$NS_{\{1,10\}}$	1	1
NS_6	$NS_{\{2,10\}}$	2	1
NS_7	$NS_{\{2,3,10\}}$	2	2
NS_8	$NS_{\{2,3\}}$	2	2
NS_9	$NS_{\{1,2,3,4\}}$	3	2
NS_{10}	$NS_{\{2,3,5,6,7\}}$	5	5
NS_{11}	$NS_{\{2,3,5,6,7\}}$	1	1

Fig. 9 System-level WCET calculation

synchronization edge could safely prevent these sequences from happening in parallel. The complete results of the MHP analysis for the example are given in the right part of Figure 9.

Step 4. Core-Level Analysis: At this is step, the static core-level analysis tool is utilized to acquire the WCET and Worst-Case Memory Accesses (WCMA) of each sequence (WCMA includes both data accesses and instruction fetches). In our example, the $WCET(NS_i)$ of each node sequence NS_i is given in the right part of Figure 9.

Step 5. Interference Analysis: According to the results of the MHP analysis and the WCMA, the possible interference delays of each sequence are calculated; this is achieved due to knowledge of the WCMA and the shared resource arbitration schemes as specified by the ADL. In our example, the interference delays $IF(NS_i)$ of each node sequence NS_i is given in the right part of Figure 9.

Step 6. Communication Analysis: Following, the communication analysis is used to determine communication delays (I.e. blocking times at *Wait* operations). The communication analysis starts from the first sequence of each core and

calculates the worst-case start time (σ_i) and end time (ϵ_i) of each sequence NS_i as follows:

$$\sigma_i = \max_{NS_j \in pred(NS_i)} (\epsilon_j) \quad \text{and} \quad \epsilon_i = \sigma_i + WCET(NS_i) + IF(NS_i) \tag{1}$$

with $pred(NS_i)$ being the predecessor sequences of NS_i in the acyclic-PPIR, considering both intra and inter-core edges. For example, the worst-case end times of NS_1 and NS_4 are $\epsilon_1 = 5$ and $\epsilon_4 = 6$. Consequently, the start of NS_2 is the maximum of the two, i.e. $\sigma_2 = 6$.

Using the start and end times of each sequence, the communication delay $COM(NS_i)$ of a sequence NS_i is:

$$COM(NS_i) = \sigma_i - \epsilon_{pred_k(NS_i)} \tag{2}$$

where $pred_k(NS_i)$ is the predecessor sequence of NS_i on the same core. Continuing the previous example, the communication delay of sequence NS_2 at the wait node W_1 is $\sigma_2 - \epsilon_1 = 1$.

Step 7. System-Level Analysis: As a final step, the static analysis tool aiT is configured (through annotations) to add the interference delays $IF(NS_i)$ and communication delays $COM(NS_i)$ for each of the sequences NS_i . With the delays added, the WCET calculation of the static analysis tool provides the SL-WCET for each core. The global SL-WCET corresponds to the maximum of the SL-WCETs of all cores.

VI. Experimental Evaluation

In our experimental evaluation, we use the ARGO tools to parallelize the model-based TAWS application and a C-implementation of the *Canny Edge Detection* algorithm. The latter is an image processing algorithm used in avionics e.g. for vision-based aircraft navigation systems such as [38]. We determine safe static WCET values for the generated code and additionally measure the average-case execution times (ACET) for the TAWS on the target platform. As a reference to compute speedup numbers for parallel programs, we use the WCET/ACET of the sequential C code before code transformations on a single core of the platform. For the TAWS, we only measure/analyze the core algorithm without routines that handle data input and output. We use pre-defined test-bench data as stimuli for the TAWS, but make sure that the data is not visible for the value analysis of the WCET analyzer. By means of pseudo-random number generation, we ensure that the test-bench inputs have enough variance to get a realistic average-case performance.

A. Target Hardware Platforms

As hardware target for our evaluations, we use a timing-predictable platform based on the InvasIC research architecture [39].The platform consists of LEON3 processor cores embedded into a WCET-customized network-on-chip (NoC). The LEON3 cores implement the SPARC-V8 RISC instruction set and are based on the intellectual property (IP)

products of Cobham Gaisler[‡‡]. Each core resides in an independent processing tile with an Advanced High-performance Bus (AHB) as a local interconnect. The AHB bus connects the core to the tile local memory (TLM) and the network adapter as shown in Figure 10. The TLM contains program code as well as local program data, which are both cached in configurable L1 data and instruction caches. While the ARGO flow generally supports instruction and data fetches from shared memories, the distributed memory structure of the evaluation platform helps to avoid unnecessary interference effects.

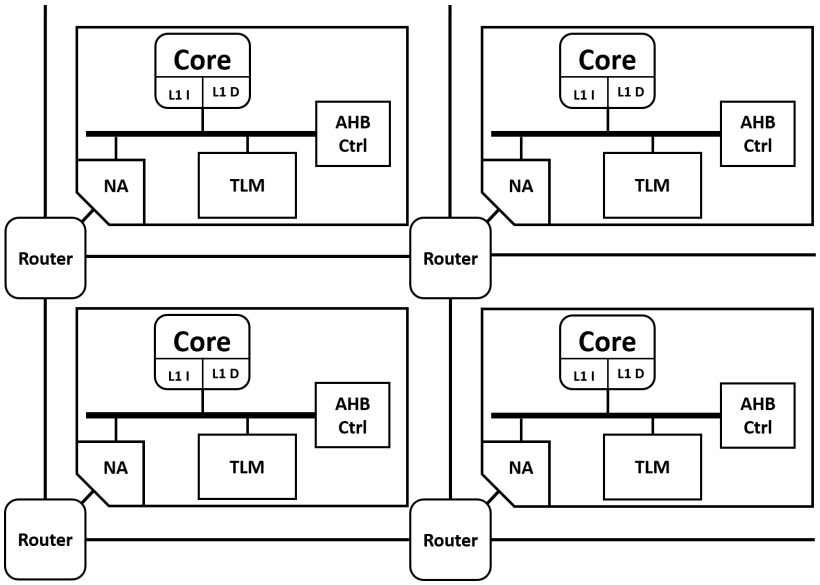


Fig. 10 System overview of the InvasIC platform

The interconnects inside the tiles are free of interference, since the LEON3 core is the only master of the local AHB bus. Communication between tiles is provided by the network adapter, which implements a register interface for message-based communication through the NoC. The NoC itself uses a scalable 2D mesh topology, consisting of packet switching routers to transmit data from the source to the destination tile. The routers use an XY-routing algorithm, which first minimizes the distance between source and destination in X-direction before routing in Y-direction. The data is subdivided into flits while traversing the NoC through so-called virtual channels. A virtual channel is an end-to-end connection between two tiles, which can be configured either for best effort (BE) or for guaranteed service (GS) traffic. To ensure real-time capability, we only use GS connections in our evaluation. By means of weighted round-robin arbitration, GS connections have guaranteed worst-case latencies/throughputs, while the maximum NoC performance can still be utilized in the absence of interference.

The InvasIC platform is implemented as an FPGA prototype on a Xilinx Virtex UltraScale+ FPGA VCU118 Evaluation Kit. For our experiments, it was configured to run at 100MHz while the TLM size was set to 512 Kbyte. The L1 instruction and data caches have each been configured as 8 Kbytes two-way associative cache with least recently used

[‡‡] `https://www.gaisler.com/leon3`

(LRU) replacement strategy. The LEON3 cores in the InvasIC design do not have a hardware floating-point unit (FPU).

B. Results

Table 1 shows the ACET and WCET in µs for the TAWS on the InvasIC platform prototype. The ARGO tools generated a parallelization that utilizes all four cores of the platform while the data fields fitted into the TLMs. The scheduling step was configured to use a WCET-aware HEFT-LA heuristic and all floating-point computations used double precision. For the first parallelization in the table, all optimizations have been enabled. The second one was generated using the same schedule, but with communication placement optimization disabled (Phase 2 of the "Data Management & Synchronization" step).

Table 1 Execution times in µs and speedups (in brackets) for the TAWS application on the InvasIC platform

Scenario	WCET	ACET
sequential code	7788 µs	672 µs
parallel code (speedup)	4410 µs (1.77)	524 µs (1.28)
parallel w/o comm. opt.	4792 µs (1.63)	641 µs (1.05)

With activated communication optimization, one iteration of the parallelized TAWS requires 314 communication operations and the individual cores accumulate in average 2114 µs worst-case synchronization delay while waiting for other cores. This is a relatively high ratio of communication to computation, which indicates that the TAWS is rather difficult to parallelize. Nevertheless, the ARGO tools achieve a speedup of 1.77 in WCET performance compared to the single-core version. With the corresponding WCET of the parallelized code, we achieve a minimum update rate of 226Hz for the TAWS algorithm on the InvasIC platform. The measured ACET is improved by the lower factor of 1.28, which can be explained by the WCET-oriented optimization strategy of the ARGO tools and the significantly lower computational load in the average case scenario. Optimizing for ACET speedups as a second criterion is not (yet) implemented in the ARGO tool-flow. This criterion would become relevant for mixed-criticality systems, where the spare time gained from an earlier completion of hard real-time computations can be used to execute non-critical tasks.

The reference scenario with disabled communication placement optimization shows that this step improves the parallel WCET by 8.7% and the ACET by 22.3%. The refined placement thus significantly contributes to the overall speedup of the communication-heavy TAWS application.

Although the target platform does not include a hardware FPU, we also ran the ARGO tools for the TAWS with the assumption that SPARC-V8 floating-point instructions are enabled. The tools generate sequential output code in this case since no beneficiary parallelization could be found. With 484µs, the resulting WCET of the sequential code with FPU instructions was significantly lower. Generally, the comparison of the ACET and WCET times in Table 1 shows a significant gap between the safe worst-case times and the measured execution times. The fact that the

sequential WCET with FPU is even lower than the ACET without FPU indicates that the software implementations of the floating-point computations are a major reason for that gap. This is not surprising since the execution times of software floating-point routines can change significantly depending on the input data. Without further information, a static WCET analyzer must assume the worst possible input, which results in relatively pessimistic WCET values compared to the ACET.

The TAWS is, on the one hand, a good use-case to evaluate the ARGO tools for applications with high communication overhead. On the other hand, benchmarks with higher memory usage are more suitable to demonstrate the handling of interference in the ARGO flow. In general, there is a trade-off between communication costs and interference costs in the presented WCET-aware parallelization approach. Less communication and synchronization results in fewer constraints for the MHP analysis, which in turn increases the interference costs, since more memory accesses may happen in parallel.

We use the complementary canny-edge-detector use-case to evaluate the ARGO flow for the other side of this trade-off. This benchmark is characterized by a lower communication overhead, but has a higher computational complexity and requires more memory. To provide sufficient memory, we extended the ADL description for InvasIC to model a platform with enabled FPU and an uncached 2GB off-chip shared memory that is connected to the AHB bus of one of the tiles. This memory is assumed to have a load-access delay of 7 cycles for the local LEON3 core, while additional delays are added for NoC transfers and interference effects. To parallelize the edge detection algorithm, the input image with 256x256 pixels (in single-precision floating representation) is initially divided into four overlapping tiles. These tiles can be processed independently before being merged into a single output image. The shared memory was used by the memory-mapping component to store the input and output images as well as several intermediate images that could not be mapped to the TLMs for space reasons. The result for this benchmark is a WCET speedup of 2.66 using four processor cores. With the image split into tiles, the parallelization requires only six communication operations but suffers from 12.5% WCET penalty[§§] caused by interference when accessing the shared memory.

VII. Conclusion

In this article, we have shown that complex tool-flows consisting of automated parallelization and Worst-Case-Execution-Time (WCET) analysis tools can be successfully used to automate the process of adapting real-time programs to multi-core target platforms. By parallelizing a model-based Terrain Awareness and Warning System (TAWS) as well as a Canny Edge Detection algorithm, we demonstrate the feasibility of this approach for both communication-intensive applications and benchmarks with shared memory usage and interference effects. The resulting WCET speedups of 1.77 for the TAWS and 2.66 for the edge detector furthermore show that hard real-time applications can significantly benefit from the use of multi-core platforms. From the successful end-to-end implementation of a parallel TAWS with

[§§]The penalty refers to the sum of interference costs for all cores relative to the sum of worst-case execution times for all cores.

model-based design principles, we can conclude that the presented tool flow, together with the proposed user-interactive workflow, can help to significantly improve the productivity in the development of hard real-time applications.

If high speedups can be achieved for a wider range of safety-critical avionics applications, the presented tools may have the potential to facilitate the adoption of both multi-core computing platforms and model-based design principles in avionics. It is part of the planned future work to investigate this potential and evaluate the approach with larger sets of use-case applications and hardware target platforms. Concerning a potential technology transfer from a research prototype to the actual utilization in avionic software development, tool qualification is a critical issue to be addressed. However, the question of qualifying the presented approach is strongly related to general research efforts for certifying multi-core systems and therefore should not be considered in isolation. Demonstrating the feasibility of WCET-aware parallelization is a significant step towards multi-core platforms in avionics, but considerable future efforts will be necessary to meet and verify all requirements for qualification.

Funding Sources

This work has been co-funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 688131 – ARGO.

References

[1] Durak, U., Becker, J. H., Hartmann, S., and Voros, N. S. (eds.), *Advances in Aeronautical Informatics : Technologies Towards Flight 4.0*, SpringerLink : Bücher, Springer, Cham, 2018. doi:10.1007/978-3-319-75058-3.

[2] Ferdinand, C., and Wilhelm, R., "Efficient and Precise Cache Behavior Prediction for Real-Time Systems," *Real-Time Systems*, Vol. 17, No. 2, 1999, pp. 131–181. doi:10.1023/A:1008186323068.

[3] Kästner, D., Schlickling, M., Pister, M., Cullmann, C., Gebhard, G., Heckmann, R., and Ferdinand, C., "Meeting Real-Time Requirements with Multi-core Processors," *Computer Safety, Reliability, and Security*, edited by F. Ortmeier and P. Daniel, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 117–131. doi:10.1007/978-3-642-33675-1_10.

[4] Hamza, R., Matthieu, M., Claire, M., Robert I., D., and Sebastian, A., "Response Time Analysis of Synchronous Data Flow Programs on a Many-core Processor," *In proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS 2016)*, ACM, 2016, pp. 67–76. doi:10.1145/2997465.2997472.

[5] Skalistis, S., and Simalatsar, A., "Near-optimal deployment of dataflow applications on many-core platforms with real-time guarantees," *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2017, pp. 752–757. doi:10.23919/DATE.2017.7927090.

[6] Rouxel, B., Derrien, S., and Puaut, I., "Tightening Contention Delays While Scheduling Parallel Applications on Multi-core Architectures," *ACM Trans. Embed. Comput. Syst.*, Vol. 16, No. 5s, 2017, pp. 164:1–164:20. doi:10.1145/3126496, URL `http://doi.acm.org/10.1145/3126496`.

[7] Rouxel, B., Skalistis, S., Derrien, S., and Puaut, I., "Hiding communication delays in contention-free execution for SPM-based multi-core architecturese," *31th Euromicro Conference on Real-Time Systems (ECRTS)*, 2019. To appear.

[8] Derrien, S., Puaut, I., Alefragis, P., Bednara, M., Bucher, H., David, C., Debray, Y., Durak, U., Fassi, I., Ferdinand, C., Hardy, D., Kritikakou, A., Rauwerda, G., Reder, S., Sicks, M., Stripf, T., Sunesen, K., ter Braak, T., Voros, N., and Becker, J., "WCET-aware parallelization of model-based applications for multi-cores: The ARGO approach," *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 286–289. doi:10.23919/DATE.2017.7927000.

[9] Alefragis, P., Theodoridis, G., Katsimpris, M., Valouxis, C., Gogos, C., Goulas, G., Voros, N., Reder, S., Kasnakli, K., Bednara, M., Müller, D., Durak, U., and Becker, J., "Mapping and Scheduling Hard Real Time Applications on Multicore Systems - The ARGO Approach," *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, edited by N. Voros, M. Huebner, G. Keramidas, D. Goehringer, C. Antonopoulos, and P. C. Diniz, Springer International Publishing, Cham, 2018, pp. 700–711. doi:10.1007/978-3-319-78890-6_56.

[10] Nowotsch, J., and Paulitsch, M., "Leveraging multi-core computing architectures in avionics," *2012 Ninth European Dependable Computing Conference*, IEEE, 2012, pp. 132–143. doi:10.1109/EDCC.2012.27.

[11] Agrou, H., Sainrat, P., Gatti, M., and Toillon, P., "Mastering the behavior of multi-core systems to match avionics requirements," *Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st*, IEEE, 2012, pp. 6E5–1. doi:10.1109/DASC.2012.6382403.

[12] Löfwenmark, A., and Nadjm-Tehrani, S., "Challenges in future avionic systems on multi-core platforms," *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, IEEE, 2014, pp. 115–119. doi:10.1109/ISSREW.2014.70.

[13] Sander, O., Bapp, F., Dieudonne, L., Sandmann, T., and Becker, J., "The promised future of multi-core processors in avionics systems," *CEAS Aeronautical Journal*, Vol. 8, No. 1, 2017, pp. 143–155. doi:10.1007/s13272-016-0228-x.

[14] Leupers, R., Aguilar, M. A., Eusse, J. F., Castrillon, J., and Sheng, W., "MAPS: A Software Development Environment for Embedded Multicore Applications," *Handbook of Hardware/Software Codesign*, 2017, pp. 917–949. doi:10.1007/978-94-017-7267-9_2.

[15] Cordes, D., Neugebauer, O., Engel, M., and Marwedel, P., "Automatic Extraction of Task-Level Parallelism for Heterogeneous MPSoCs," *2013 42nd International Conference on Parallel Processing*, 2013, pp. 950–959. doi:10.1109/ICPP.2013.113.

[16] Cordes, D., Marwedel, P., and Mallik, A., "Automatic parallelization of embedded software using hierarchical task graphs and integer linear programming," *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ACM, 2010, pp. 267–276. doi:10.1145/1878961.1879009.

[17] Stripf, T., Oey, O., Bruckschloegl, T., Becker, J., Rauwerda, G., Sunesen, K., Goulas, G., Alefragis, P., Voros, N. S., Derrien, S., Sentieys, O., Kavvadias, N., Dimitroulakos, G., Masselos, K., Kritharidis, D., Mitas, N., and Perschke, T., "Compiling Scilab to high performance embedded multicore systems," *Microprocessors and Microsystems*, Vol. 37, No. 8, Part C, 2013,

pp. 1033 – 1049. doi:10.1016/j.micpro.2013.07.004, URL http://www.sciencedirect.com/science/article/pii/S014193311300094X, special Issue on European Projects in Embedded System Design: EPESD2012.

[18] Ungerer, T., Bradatsch, C., Frieb, M., Kluge, F., Joerg, Mische, Stegmeier, A., Jahr, R., Gerdes, M., Zaykov, P., Matusova, L., Li, Z. J. J., Petrov, Z., Böddeker, B., Kehr, S., Regler, H., Hugl, A., Rochange, C., Ozaktas, H., Cassé, H., Bonenfant, A., Sainrat, P., Lay, N., George, D., Broster, I., Quiñones, E., Panić, M., Abella, J., Hernández, C., Cazorla, F. J., Uhrig, S., Rohde, M., and Pyka, A., "Experiences and Results of Parallelisation of Industrial Hard Real-time Applications for the parMERASA Multi-core," *3rd Workshop on High-performance and Real-time Embedded Systems (HiRES 2015), Amsterdam, the Netherlands*, 2015.

[19] Milind Girkar and Constantine D. Polychronopoulos, "The hierarchical task graph as a universal intermediate representation," *International Journal of Parallel Programming*, Vol. 22, No. 5, 1994, pp. 519–551. doi:10.1007/bf02577777.

[20] Ozaktas, H., Rochange, C., and Sainrat, P., "Automatic WCET Analysis of Real-Time Parallel Applications," *13th International Workshop on Worst-Case Execution Time Analysis*, OpenAccess Series in Informatics (OASIcs), Vol. 30, edited by C. Maiza, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013, pp. 11–20. doi:10.4230/OASIcs.WCET.2013.11, URL http://drops.dagstuhl.de/opus/volltexte/2013/4118.

[21] Potop-Butucaru, D., and Puaut, I., "Integrated Worst-Case Execution Time Estimation of Multicore Applications," *13th International Workshop on Worst-Case Execution Time Analysis*, OpenAccess Series in Informatics (OASIcs), Vol. 30, edited by C. Maiza, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013, pp. 21–31. doi:10.4230/OASIcs.WCET.2013.21, URL http://drops.dagstuhl.de/opus/volltexte/2013/4119.

[22] Kelter, T., "WCET analysis and optimization for multi-core real-time systems," Ph.D. thesis, Technischen Universität Dortmund, 2015. doi:10.17877/DE290R-7209.

[23] Kelter, T., Harde, T., Marwedel, P., and Falk, H., "Evaluation of resource arbitration methods for multi-core real-time systems." *WCET*, 2013, pp. 1–10. doi:10.4230/OASIcs.WCET.2013.1.

[24] Becker, M., Dasari, D., Nicolic, B., Akesson, B., Nélis, V., and Nolte, T., "Contention-free execution of automotive applications on a clustered many-core platform," *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on*, IEEE, 2016, pp. 14–24. doi:10.1109/ECRTS.2016.14.

[25] Reder, S., Masing, L., Bucher, H., ter Braak, T., Stripf, T., and Becker, J., "A WCET-aware parallel programming model for predictability enhanced multi-core architectures," *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 943–948. doi:10.23919/DATE.2018.8342145.

[26] Lefeuvre, T., Fassi, I., Cullmann, C., Gebhard, G., Kasnakli, E. K., Puaut, I., and Derrien, S., "Using polyhedral techniques to tighten WCET estimates of optimized code: A case study with array contraction," *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2018, pp. 925–930. doi:10.23919/DATE.2018.8342142.

[27] Stürmer, I., Conrad, M., Fey, I., and Dörr, H., "Experiences with model and autocode reviews in model-based software development," *Proceedings of the 2006 international workshop on Software engineering for automotive systems*, ACM, 2006, pp. 45–52. doi:10.1145/1138474.1138483.

[28] Campbell, S. L., Chancelier, J.-P., and Nikoukhah, R., *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*, Springer, 2010. doi:10.1007/0-387-30486-X.

[29] Durak, U., Müller, D., Becker, J., Voros, N. S., Alefragis, P., Stripf, T., Agnel, P.-A., Rauwerda, G., and Sunesen, K., "Model-based Development of Enhanced Ground Proximity Warning System for Heterogeneous Multi-Core Architectures," *23. Symposium Simulationstechnik (ASIM 2016)*, ASIM, 2016, pp. 159–167.

[30] Durak, U., Müller, D., Möcke, F., and Koch, C. B., "Modeling and simulation based development of an enhanced ground proximity warning system for multicore targets," *Proceedings of the Model-driven Approaches for Simulation Engineering Symposium*, SCS, 2018, p. 4.

[31] Koch, C. B., Durak, U., and Müller, D., "Simulation-based verification for parallelization of model-based applications," *Proceedings of the 50th Computer Simulation Conference*, SCS, 2018, p. 10.

[32] Ulbig, P., Müller, D., Torens, C., Insaurralde, C. C., Stripf, T., and Durak, U., "Flight Simulator-Based Verification for Model-Based Avionics Applications on Multi-Core Targets," *AIAA Scitech 2019 Forum*, AIAA, 2019, p. 1976. doi:10.2514/6.2019-1976.

[33] Durak, U., Stürmer, I., Pawletta, T., and Mahmoodi, S., "Quality Assessment and Quality Improvement in Model Engineering," *Model Engineering for Simulation*, Elsevier, 2019, pp. 209–231. doi:10.1016/B978-0-12-813543-3.00010-X.

[34] Gogos, C., Valouxis, C., Alefragis, P., Goulas, G., Voros, N., and Housos, E., "Scheduling independent tasks on heterogeneous processors using heuristics and Column Pricing," *Future Generation Computer Systems*, Vol. 60, 2016, pp. 48 – 66. doi:10.1016/j.future.2016.01.016, URL http://www.sciencedirect.com/science/article/pii/S0167739X16000297.

[35] Topcuoglu, H., Hariri, S., and Min-You Wu, "Task scheduling algorithms for heterogeneous processors," *Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99)*, 1999, pp. 3–14. doi:10.1109/HCW.1999.765092.

[36] Emeretlis, A., Theodoridis, G., Alefragis, P., and Voros, N., "Static Mapping of Applications on Heterogeneous Multi-Core Platforms Combining Logic-Based Benders Decomposition with Integer Linear Programming," *ACM Trans. Des. Autom. Electron. Syst.*, Vol. 23, No. 2, 2017, pp. 26:1–26:24. doi:10.1145/3133219, URL http://doi.acm.org/10.1145/3133219.

[37] Ferdinand, C., and Heckmann, R., "aiT: Worst-case execution time prediction by static program analysis," *Building the Information Society*, 2004, pp. 377–383. doi:10.1007/978-1-4020-8157-6_29.

[38] Ganguli, S., and Krishnaswamy, K., "Vision based navigation and guidance system," , Feb. 1 2011. US Patent 7,881,497.

[39] Henkel, J., Herkersdorf, A., Bauer, L., Wild, T., Hübner, M., Pujari, R. K., Grudnitsky, A., Heisswolf, J., Zaib, A., Vogel, B., Lari, V., and Kobbe, S., "Invasive manycore architectures," *17th Asia and South Pacific Design Automation Conference*, 2012, pp. 193–200. doi:10.1109/ASPDAC.2012.6164944.