WCET-Aware Parallelization of Model-Based Applications for Multi-Cores: the ARGO Approach

Steven Derrien*, Isabelle Puaut*, Panayiotis Alefragis†, Marcus Bednara‡, Harald Bucher§, Clément David¶, Yann Debray¶, Umut Durak ‖, Imen Fassi*, Christian Ferdinand**, Damien Hardy*, Angeliki Kritikakou*, Gerard Rauwerda††, Simon Reder§, Martin Sicks**, Timo Stripf§, Kim Sunesen††, Timon ter Braak††, Nikolaos Voros†, Jürgen Becker§
*Université de Rennes I (UR1)
†Technological Educational Institute of Western Greece (TWG)
‡Fraunhofer IIS (IIS)
§Karlsruhe Institute of Technology (KIT)
¶Scilab Enterprises (SCILAB)
‖Deutsches Zentrum fur Luft-Und Raumfahrt (DLR)
**Absint Angewandte Informatik GmbH (ABSINT)
††Recore Systems B.V. (RS)

***Abstract*—Parallel architectures are nowadays not only confined to the domain of high performance computing, they are also increasingly used in embedded time-critical systems. The ARGO H2020 project[1] provides a programming paradigm and associated tool flow to exploit the full potential of architectures in terms of development productivity, time-to-market, exploitation of the platform computing power and guaranteed real-time performance. In this paper we give an overview of the objectives of ARGO and explore the challenges introduced by our approach.**

I. Introduction

Increased performance at reduced cost, while maintaining real-time reliability and programmability, are key demands in several domains such as aerospace and automotive industries. Driven by the technology restrictions in chip design, the end of exponential growth of clock speeds, and the increasing request for computing performance, even the most powerful embedded processors cannot cope with these demands, leading to the increasing use of multi- and many-core architectures.

Embedded systems interact with their environment. Thus, in general they must react within time limits under an appropriate level of safety. These *real-time* requirements limit the processor and communication hardware to *deterministic* components. In the past, time-critical applications were typically mapped to deterministic single core processors. Single-core processors are not capable anymore of offering sufficient performance for the constantly increasing computational demands. While in non-critical environments the usage of multi-core processors with complex cores is state-of-the-art, these complex processors cannot be used safely for time-critical applications because they are not deterministic enough to meet strict timing guarantees. Solving the computational demands of next generation time-critical applications on deterministic components is a major challenge of this decade.

For time-critical applications, deadlines are guaranteed to be met in all circumstances by calculating the application's worst-case execution time, or WCET [1]. To be *safe*, WCET estimates have to be higher than or equal to any possible execution time. In addition, to be useful they have to be as close as possible to the actual WCET (*tightness*). The prerequisite for calculating *tight* WCET is a deterministic hardware architecture that avoids dynamic, hard-to-predict calculations at run-time. Moreover, accesses to shared resources, such as main memory must be organized to guarantee a worst-case access time under any circumstances. The challenge here is to avoid over-pessimistic WCET estimates for multi-cores.

Furthermore, parallel programming of embedded applications for multiprocessors suffers from a complex programming process, often reserved to High Performance Computing (HPC) experts. The problems are the absence of standards for parallel programming, the indeterminism of the architectures during debugging, deadlocks, etc. This makes parallel programming notoriously much harder, error-prone, time-consuming and in consequence costly compared to sequential programming. The usage of model-based development techniques, combined with automatic parallelization, is capable of hiding most of these problems to the end users, but suffers from a lack of suitable and generic parallelization tool-chains.

The ARGO proposal intends to provide a cross-layer programming approach for exploiting the full potential of next generation heterogeneous parallel embedded systems under real-time constraints. The ARGO cross-layer programming combines the following technologies in a holistic approach:

- *Model-based development and testing*, to increase productivity, shorten time-to-market and reduce porting efforts;
- *Cross-layer programming user interface* and *WCET-aware automatic parallelization*, to improve the worst-case execution time and reduce the gap between the worst-case and average-case execution time;
- *WCET-analysis for heterogeneous multi- and many-core architectures* to provide strict upper bounds of the system's worst-case execution time.

This paper presents the vision and the technical challenges of ARGO. Section II first gives an overview of the ARGO tool-chain. Section III then lists the main challenges and how we plan to address them. Section IV describe the use cases that are being developed to validate the ARGO approach.

[1]ARGO (http://www.argo-project.eu/) is funded by the European Commission under Horizon 2020 Research and Innovation Action, Grant Agreement Number 688131.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

II. Overview of the ARGO programming tool chain

In ARGO, we address the challenge of obtaining tight WCET estimates for multi-cores by combining WCET calculation with an automatic parallelization of application models. Thereby, the application models are automatically parallelized, mapped and scheduled to the multi-core hardware. During the scheduling phase, parallel accesses to shared resources, such as communication infrastructure and shared memory, are considered. At any point in time, all shared resource contenders are known and their number is reduced during parallelization to avoid overly pessimistic WCET estimates. The ARGO design workflow is depicted in Figure 1 and described in the following subsections.

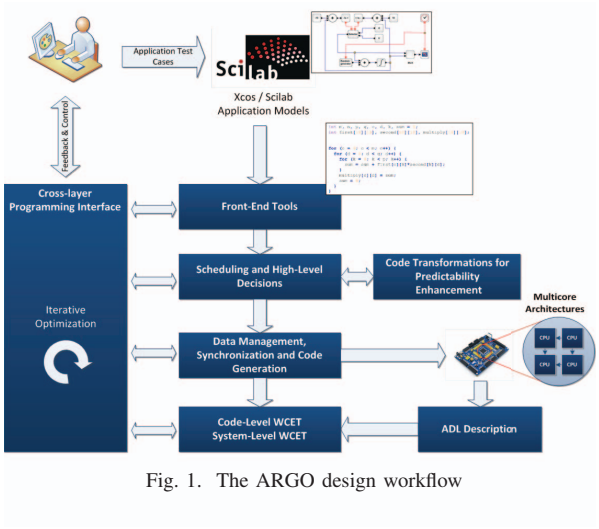

Fig. 1. The ARGO design workflow

A. Model Based Design specifications

In ARGO, the end users describe their applications using a combination of dataflow modeling, using the open-source Xcos modeling framework, and high-level programming using Scilab[2]. To provide an extensible framework, the behavior of all Xcos components used in ARGO is also described in the Scilab language. Thanks to this approach the end-users can both use a modeling approach and a high level functional specification of their applications. The supported hardware platforms are also specified using a model-based approach thanks to the ARGO Architecture Description Language (ADL). The proposed ADL provides all the information required by the tool-chain (processors, memory, interconnect, etc.) to calculate WCETs.

B. Program analysis and task level parallelization

The Xcos/Scilab models are then compiled to an intermediate program representation (IR) based on a subset of the C language. This IR is used as input by the GeCoS source-to-source transformation framework [2], which performs several predictability enhancing program transformations (scratchpad management for data, predictability oriented task parallelism extraction through loop transformations, etc.).

Afterwards, a task extraction stage is applied to the program, from which we obtain a Hierarchical Task Graph (HTG) corresponding to the program. In a HTG, loops are enclosed in an additional hierarchy level, resulting in a hierarchy of acyclic task graphs. In a HTG, tasks dependencies embed information on the variables and the buffers that need to be communicated between tasks, while task nodes include additional information on possible shared resource accesses (list of shared resources, and worst case number of accesses). The HTG obtained from the input program is then mapped on the target platform during a scheduling/mapping stage which computes an optimized schedule and mapping of tasks to processors.

[2]http://www.scilab.org/

C. Parallel program model construction

The result of the scheduling/mapping stage is used to transform the initial program representation into an explicit parallel program model, in which the synchronizations are made explicit, and the final memory address mapping of the variables and the buffers is obtained. The result of this stage is an explicitly parallel IR, which is used to run the system-level WCET analysis, and generate C code following the WCET-aware programming model for the target platforms.

D. Code-level and system-level WCET analysis

Code-level and system-level WCET analysis jointly calculate the multi-core WCET for the target architectures. Our approach relies on the use of an explicitly parallel program representation, in which information pertaining to core/code-level and system-level WCET is exposed. Code-level WCET estimation calculates the isolated WCET of code fragments on one core, regardless of the core fragments assigned to the other cores. This stage ignores the cost of resource contentions, which is handled by the system-level WCET estimation stage. System-level WCET estimation builds on the parallel program representation to precisely identify resource conflicts. This is achieved through (i) a static analysis that determine as accurately as possible if several code snippets may happen in parallel and (ii) a cost model of the interference derived from the platform abstract models.

E. Iterative optimization through cross layer programming

To solve the unavoidable phase ordering problem faced by such a tool-chain, WCET information is fed back to the previous compilation phases to enable an iterative optimization of the parallelization process. This process is managed through a cross-layer programming interface, in which a graphical interface exposes to end users at various abstraction levels the complex optimization decisions made by the ARGO tool-chain. The goal of this interface is to allow the end users, who may not be compiler/parallelization experts, to interact with the compilation process. Application bottlenecks can be identified and the artifacts hindering an efficient parallelization can be outlined. Furthermore, the end users can control and influence the complex parallelization decisions. Their global view and application know-how can be brought into good use to enable an efficient parallelization.

III. ARGO challenges

A. Challenges in design productivity

Model based design techniques are now widely spread in industry, and in particular in aerospace, automotive, and industrial automation to face the rising complexity of systems and platforms. In a model based design approach, the development process is managed from an abstract point of view, in which domain specific knowledge is exposed in the flow. Raising the level of abstraction of the design eases specification, but also

simplifies the validation of the system behavior thanks to the use of specialized simulation and/or validation tools. Once the model has been validated it must be implemented on the target platform. Here again, model based design approaches offer a significant advantage thanks to code generation tools, which drastically simplify the implementation stage.

However, existing tool-chains lack proper support for multi-core systems. Although some tool-chains (such as ASCET) support multi-core platforms, they do not perform automatic parallelization. More importantly they currently do not address real-time constraints. The ARGO approach aims at offering an enabling technology to use multi- and many-core systems in a model-based design workflow for real-time application.

B. Challenges in architecture design

A fundamental requirement for obtaining WCET estimates using WCET analyzers, such as AbsInt's aiT [3], is that the timing behavior of the processor is *predictable*. Even in single-core processors, predictability is compromised by speculative hardware mechanisms such as caches or branch prediction. On multi-core processors, hardware resources are shared for cost, energy, and communication reasons. The additional interferences due to concurrent accesses to such shared resources have to be considered. Even if the sharing of a resource only slightly increases the actual execution time of a task, it might be difficult for a static analysis to provide an upper bound of the increase, because an exhaustive enumeration of architectural states is practically infeasible. In ARGO, we follow the following design guidelines for predictable multi-core architectures:

- The multi-core architecture must be composed of time-predictable processors. Scratchpad memories are preferred to caches because they enable more precise WCET estimation. Any hard-to-predict mechanisms (dynamic branch prediction, prefetch, write-buffers, cache coherence) should be avoided.
- The number of shared resources between cores should be minimized. Shared last-level caches between cores should also be avoided or should be partitioned, as they lead to very pessimistic WCET estimates.
- The target architecture should use a predictable interconnect system, for which it is possible to obtain (i) worst-case delay for gaining access to the interconnect; (ii) worst-case delay for copying/getting the information, once access to the interconnect is granted.
- Fully *timing compositional* architecture. A system is said to be time compositional if "the contribution of individual components to the overall system's timing can be considered separately, and there exists a function to combine the components' respective timings".

C. Challenges in WCET estimation

Accurate WCET analysis for multi-cores is known to be challenging, because of concurrent accesses to shared resources. Since it is impossible in general to guarantee the absence of resource conflicts during execution, current WCET techniques either produce pessimistic WCET estimates or constrain the execution to enforce the absence of conflicts, at the price of a significant hardware under-utilization.

In addition, the large majority of existing works consider that the platform workload consists of independent tasks. As parallel programming is the most promising solution to improve performance, we envision that within only a few years from now, real-time workloads will evolve toward parallel programs. The WCET behaviour of such programs is challenging to analyze because they consist of *dependent* tasks interacting through complex synchronization/communication mechanisms. So far, very few works have addressed this problem. A notable exception is the parMERASA European project, which aimed, among others, at integrating synchronization costs in the WCET for parallel programs. However, the experiments carried out in the project [4] have shown that manually parallelized programs are difficult to analyze and may lead to very pessimistic WCET. The reasons behind this are twofold: (i) in order to obtain an accurate WCET estimate of the program, tools need to be able to reason about the interactions between tasks. Without a high level knowledge of the parallelization scheme, performing such an analysis is not possible; (ii) Parallel programs are usually written by HPC experts, who aim at improving average performance, and often ignore predictability issues. This results in parallel programs whose high level behavior is even more difficult to analyze. As a consequence, we believe that the only viable solution to address the parallel WCET problem is to bring together parallelizing compilers and WCET analysis technologies into a single flow. We expect that our approach will advance state-of-the art WCET techniques by:

1) Defining a model offering a high-level view of the behavior of parallel programs, enabling a precise estimation of shared resource conflicts.
2) Exploring the interactions between parallelizing compilers and WCET estimations to propose predictable parallelization techniques (e.g., optimize programs avoiding shared resource contentions).

Parallelizing a real-time application on a multi-core involves a static scheduling and mapping stage. Such a problem is known to be a challenging (NP-hard) combinatorial optimization problem, and a large body of research work have studied this problem in the context of embedded multi-core, either from an average performance or worst case performance point of view [5]. As far as the latter is concerned, most of the existing approaches try to find an optimal parallel schedule/mapping while avoiding shared resource conflicts between tasks. In the ARGO project, we aim at exploring more subtle trade-off thanks to a very fine grain task decomposition. Because this decomposition inevitably leads to a combinatorial explosion, we envision an approach using a combination of exact techniques and advanced heuristics.

Efficient and predictable implementations also requires other program transformations/optimizations that are not directly related to parallelization on multi-core. A lot of research has been done in this direction in the past mainly for sequential code: (i) allocation of code and data in a predictable manner, using either cache locking or WCET-directed management of scratchpad memory [6]; (ii) Optimization of bus schedules to minimize WCETs [7]; (iii) Code transformations to allow parallelism between communications and computations [8]. Despite this large effort on sequential WCET-oriented optimizations, only a few works have addressed the problem of WCET-oriented optimizations in parallelizing compilers.

The choice of the task extraction strategy in parallelizing compilers has a huge impact on the program task graph, and hence on the results of the scheduling/mapping stage. Although many parallelizing transformations exist, they all target

average case performance. They must therefore be revisited in the context of performance predictability. For example, there exist optimizations that are not widely used because they involve redundant computation [9] or complex control code [10] whose overhead annihilate average case performance benefits. They may however happen to be perfectly viable and relevant in a predictable performance context.

IV. Overview of ARGO use cases

To validate the ARGO approach, the tool-chain will be validated against two use cases, developed by the aerospace and industry automation experts DLR[3] and Fraunhofer IIS[4]. We will also use two distinct multi-core platforms from Karlsruhe Institute of Technology[5] and Recore Systems[6] to demonstrate the flexibility of the ARGO tool-chain. The overall objectives of the use cases is to evaluate the benefits of the ARGO approach in terms of productivity and also in performance and predictability improvements.

A. Aerospace use case

In the aerospace domain, emerging flight control applications are becoming more and more computationally demanding. Two systems will be considered for implementation: Enhanced Ground Proximity Warning System (EGPWS) and Wake Encounter Avoidance and Advisory System (WEAA). EGPWS is already used in current aircrafts. It provides alerts and warnings for obstacle and terrain along the flight path. EGPWS combines high resolution terrain databases, GPS and other sensors to provide feedback to pilots. WEAA is a more advanced flight system concept which is not yet commercially available. WEAA provides guidance for tactical small-scale evasion from wake vortices to avoid possibly hazardous wake encounters. WEAA predicts wake vortices, performs conflict detection and generate evasion trajectories.

B. Industrial Image Processing use case

Current trends in Industrial automation and inspection tasks involve many innovative sensing technologies, such as polarization, time-of-flight or multispectral image sensors. Such advanced sensing technology relies on computationally demanding data processing algorithms, which are challenging to implement, especially when hard-real-time constraints must be met as for in-line inspection systems. Here again, the ability to automatically generate and evaluate parallel implementations on multi-cores is a key asset for designers. In ARGO, the use case will be considered for implementation is the POLKA polarization camera, already available as a product. POLKA uses a novel sensor that measures the polarization of light to detect residual stress in glass containers.

C. Target architectures

The application use cases will target embedded multi- and many-core architectures from academia and industry. The target architectures serve a threefold purpose: implementation of application use cases, demonstration and verification of ARGO tool-chain, and research and development of programmable customized and time-critical embedded computing architectures.

Recore and KIT will both provide their heterogeneous many-core systems. Recore will provide a flexible heterogeneous IP agnostic many-core platform[7] implementing a distributed multi-board system including the Xentium processor [11] and supporting more than hundred processors and accelerator cores. The platform will be prototyped on dedicated FPGA boards.

KIT will provide a heterogeneous tile-based many-core architecture that consists of compute tiles featuring either multicore processor sub-systems based on the Leon3 processor[8], or custom application accelerators. The tile level interconnect will be based on the invasive Network–on-chip (iNoC) [12], which is capable to manage data transfers effectively between hundreds or even thousands of tiles. The iNoC provides bandwidth and latency guarantees, which is required for accurate system-level WCET analysis. The platform will be implemented on dedicated FPGA boards.

References

[1] 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, and P. Stenström, "The worst-case execution-time problem – overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, pp. 36:1–36:53, May 2008.

[2] A. Floch, T. Yuki, A. E. Moussawi, A. Morvan, K. Martin, M. Naullet, M. Alle, L. L'Hours, N. Simon, S. Derrien, F. Charot, C. Wolinski, and O. Sentieys, "Gecos: A framework for prototyping custom hardware design flows," in *13th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2013, Eindhoven, Netherlands, September 22-23, 2013*, pp. 100–105, 2013.

[3] C. Ferdinand and R. Heckmann, "aiT: worst case execution time prediction by static program analysis," in *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22-27 August 2004, Toulouse, France*, pp. 377–383, 2004.

[4] H. Ozaktas, C. Rochange, and P. Sainrat, "Minimizing the cost of synchronisations in the WCET of real-time parallel programs," in *17th International Workshop on Software and Compilers for Embedded Systems, SCOPES '14, Sankt Goar, Germany, June 10-11, 2014*, pp. 98–107, 2014.

[5] H. Ding, Y. Liang, and T. Mitra, "Shared cache aware task mapping for WCRT minimization," in *18th Asia and South Pacific Design Automation Conference, ASP-DAC 2013, Yokohama, Japan, January 22-25, 2013*, pp. 735–740, 2013.

[6] S. Chattopadhyay and A. Roychoudhury, "Static bus schedule aware scratchpad allocation in multiprocessors," in *Proceedings of the ACM SIGPLAN/SIGBED 2011 conference on Languages, compilers, and tools for embedded systems, LCTES 2011, Chicago, IL, USA, April 11-14, 2011*, pp. 11–20, 2011.

[7] T. Kelter, H. Borghorst, and P. Marwedel, "WCET-aware scheduling optimizations for multi-core real-time systems," in *XIVth International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2014, Agios Konstantinos, Samos, Greece, July 14-17, 2014*, pp. 67–74, 2014.

[8] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for cots-based embedded systems," in *17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011, Chicago, Illinois, USA, 11-14 April 2011*, pp. 269–279, 2011.

[9] W. Pugh and E. Rosser, "Iteration Space Slicing and Its Application to Communication Optimization," in *Proceedings of the 11th International Conference on Supercomputing*, ICS '97, (New York, NY, USA), pp. 221–228, ACM, 1997.

[10] M. Griebl, P. Feautrier, and C. Lengauer, "Index set splitting," *Int. J. Parallel Program.*, vol. 28, pp. 607–631, Dec. 2000.

[11] Recore Systems, "Xentium® DSP Core." www.recoresystems.com/technology/, 2016.

[12] J. Heißwolf, R. König, and J. Becker, "A Scalable NoC Router Design Providing QoS Support Using Weighted Round Robin Scheduling," in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pp. 625–632, July 2012.