Extended Pattern-based Parallelization Approach for Hard Real-Time Systems and its Tool Support

Alexander Stegmeier, Martin Frieb, Theo Ungerer

Systems and Networking
Department of Computer Science
University of Augsburg
86135 Augsburg, Germany
{alexander.stegmeier,martin.frieb,ungerer}@informatik.uni-augsburg.de

Abstract: The transformation of sequential legacy code to parallel applications is hard, especially when timing requirements have to be met. There exists a systematic parallelization approach dealing with this topic. Based on practical experience, we extend it and present our modifications. Our extensions comprise an additional phase dealing with implementation details and another one for quality assurance. Its results may be used to further improve the parallel program. Moreover, we propose tool support which further facilitates the parallelization process.

1 Introduction

It is hard to parallelize a sequential legacy program [MBC11], especially when it has to meet hard real-time requirements. These programs have to finish their execution within a specific time interval (deadline). This is assured by estimating the worst case execution time (WCET)[1] [WEE$^+$08].

The parallelization approach by Jahr et al. [JGU13b, JGU13a] was developed for the parallelization of legacy sequential hard real-time applications. It was applied on several industrial applications in the parMERASA project [UBG$^+$13, UBG$^+$15]. Following this experience, we extend the approach. Our modifications are an additional phase for implementation and another one for checking functional and timing constraints. Since Jahr et al. do not describe any supporting tools and it seems that everything has to be done manually, we give a brief overview on existing and planned tools we propose for assisting the parallelization.

The structure of this paper is as follows: the next section presents related work and the original approach. In section 3, we describe our extensions. Supporting tools are characterized in section 4. Finally, the paper is concluded and an outlook is given in section 5.

[1]The execution time which might occur when running the longest possible path in the program. For applications without real-time requirements, the execution time is also an applicable measure.

2 Related Work and the Original Approach

Well-known parallelization approaches in high-performance computing are the *parallel pattern language* by Mattson et al. [MSM04, MMS99] and the *PCAM approach* by Foster [Fos95]. Inspired by those, Jahr et al. introduced a pattern-supported parallelization approach [JGU13b], which we focus on. It is platform independent and applicable for hard real-time systems [JGU13a]. In the remainder, we will refer to it as *Jahr-approach*.

This approach mainly consists of two phases which are called Reveal Parallelism and Optimize Parallelism. In the first phase (**Reveal Parallelism**), the transition from the existing source code to a model takes place, which is an extended UML2 activity diagram called activity and pattern diagram (APD). The phase is characterized by extracting segments for parallel execution and creating an APD with a high degree of parallelism. Thereby, the usage of the operators fork and join is not allowed anymore. Instead, an additional type of activity node is provided which represents parallel design patterns (PDPs)[2]. These PDPs are the only way to introduce parallelism to the program. They are taken from a pattern catalogue. In the parMERASA project, a pattern catalogue containing only timing-analysable PDPs was assembled [GJU13]. Thereby, if the sequential program is timing-analysable and only these timing-analysable PDPs are utilized to introduce parallelism to the program, the resulting parallel program will also be timing-analysable[3] [JGU$^+$14]. As a further basis for the second phase, the WCETs of the individual segments have to be determined and the dependencies between them are collected.

The goal of the second phase (**Optimize Parallelism**) is to optimize the APD for the target platform and implement the changes in the source code. At the optimization, several optimized APDs form a Pareto front in terms of minimizing their overall WCET and number of threads as well as the number of shared variables to be synchronized. Thereby, the overall WCET estimation is composed of WCET estimations of the code segments extracted in the first phase. These segments represent sequential sections in the input APD. The optimized APDs are generated by re-arranging the segments. Synchronization between the segments may only be estimated. One of the APDs from the Pareto front has to be chosen and is basis for implementing the parallel code. An example of a generated Pareto front can be seen in Figure 1. It shows the relation between several APDs in terms of their overall WCET and the number of applied threads. While the Jahr-approach does not go into detail at the implementation, we extended the approach with a third phase dealing with the implementation, see details in section 3.

In the parMERASA project [UBG$^+$13], the Jahr-approach was applied successfully in automotive, avionics [PQnZ$^+$14] and construction machinery [JFG$^+$14, JFGU14] domains. Four legacy hard real-time applications were considered: motor injection, collision avoidance, stereo navigation and the control program of a foundation crane. All of these applications were written in C. Parallelization results including WCET speedups are provided in [UBG$^+$15].

An alternative to our parallelization approach could be automatic parallelization. It is

[2]PDPs are a textual description of situations where parallelism could be applied.

[3]The targeted multi-core platform and its corresponding system software has to be timing-analysable, too.

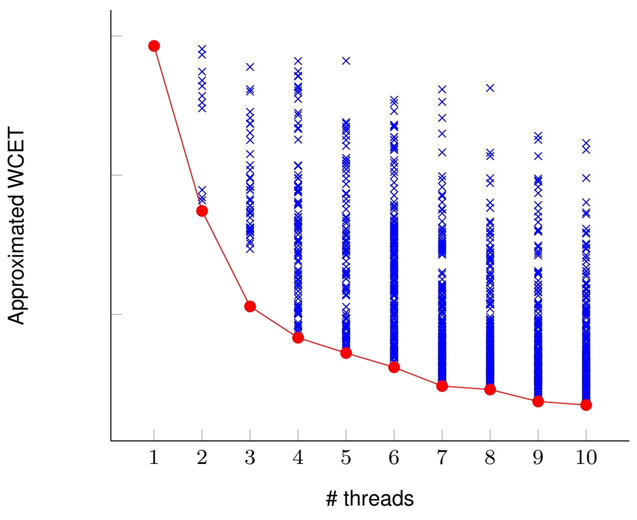

Figure 1: Optimization for minimal approximated WCET and minimal number of threads. Each $\times$ represents an APD and $\circ$ are optimal APDs belonging to the Pareto front.

researched e.g. by Cordes et al. [CMM10] or Kempf et al. [KVP11]. However, these approaches are limited to specific situations the tools are able to detect automatically. Situations fitting only roughly will not be taken into account. Therefore, chances for parallel execution may be missed. However, these tools may be utilized as assistance when trying to find situations for parallelism in the first phase.

3 Extended Parallelization Approach

Following the practical experience made in the parMERASA project, we extended the Jahr-approach. Thereby, our goal is to remain platform independent and our extended approach may also be applied on industrial embedded real-time applications.

Figure 2 illustrates how the extended parallelization approach works: **Phase I** (Revealing Parallelism) proceeds like described in the Jahr-approach, which exposes PDPs to facilitate timing analysis. Therefore, the code has to be analysed according to its data dependencies and control flow. Based on these analyses, parallel segments of code can be determined and PDPs applied to build an APD. Furthermore, the parallel segments in code have to be annotated to get a connection between code and model level.

In the Jahr-approach, the optimization as well as the implementation take place in phase II (Optimizing Parallelism). For a clear separation of concerns, we split this phase into optimization (phase II) and implementation (phase III) to focus on their specific issues. Now, **phase II** represents the model level, taking the APD of phase I, the WCETs of the segments and the list of dependencies as input and computing a set of optimized APDs.

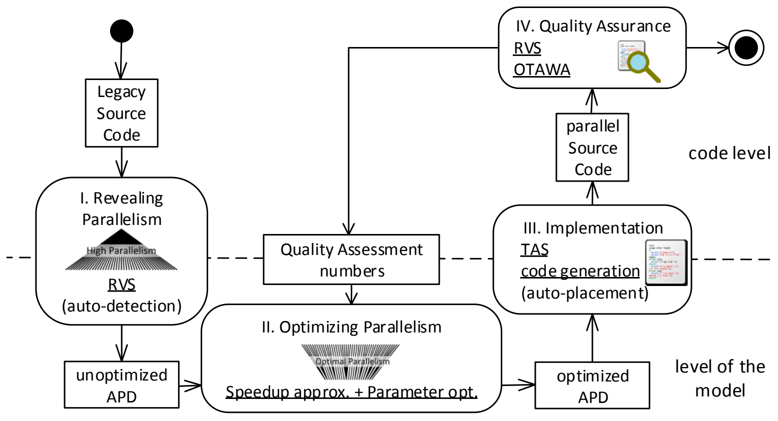

Figure 2: The four phases of the extended parallelization approach are I. Revealing Parallelism, II. Optimizing Parallelism, III. Implementation and IV. Quality Assurance. There is also a refinement loop with a feedback path from Quality Assurance to Optimizing Parallelism which is utilized to perform further optimization. Existing tools are underlined and ideas for future tools appear in parentheses.

Thereby, several variations of APDs are built. The basis for each of them is the input APD, which obtains a high degree of parallelism. The generation of the models takes place by turning on and off some of the PDPs of the basis APD. Afterwards, the generated APDs are evaluated using the estimated WCETs. The results are compared to find an optimal trade-off between minimizing the overall WCET, the synchronization effort and the applied number of threads. Finally, one APD variation is chosen from the resulting Pareto front for the implementation in phase III.

We share the opinion that the implementation is not trivial and should be considered as extra phase requiring own tool-support. In concrete, this new **phase III** includes the integration of synchronization and parallel execution in code. It takes the optimized APD of phase II and modifies the sequential legacy code according to this model. Therefore, the location of each PDP in the APD must be identified in the sequential code utilizing the annotations of phase I. The corresponding code is replaced by an implementation for parallel execution of these segments. Additionally, all synchronization points have to be identified and implemented in code. The result is source code executable on a parallel platform. Phase III leads to a clear separation of the model and code level.

In the original Jahr-approach, there is neither a check whether the deadlines hold in the parallel program, nor one for the functional correctness. Thus, we see the need of an additional **phase IV** called Quality Assurance considering these aspects and to adapt the parallel program if neccessary. The validation of functional correctness takes place by comparing the results and a defined set of variables of the sequential and parallel execution. Thereby, the code coverage is observed to ensure sufficient testing. If functional failures occur, one has to check phase I and phase III looking for possible mistakes[4]. The evaluation

[4]In phase II, it is only decided which program parts (defined by PDPs) will be executed in parallel. Therefore, no functional failures may occur.

of the parallel code's timing behavior is done by an WCET analysis, which concretely considers the overhead caused by the interaction of the parallel segments. The new results can be compared to the timing requirements.

In addition to the four phases there is a refinement loop to enable the optimization of the parallel code based on the results of phase IV. It enables the step back to model level (phase II) if the numbers do not meet the expectations (e.g. the WCET is too high). This loop involves the phases Optimize Parallelism, Implementation and Quality Assurance and utilizes the quality assurance numbers (mainly the calculated WCET) to gain a more realistic estimation of the overall WCETs in phase II. Therefore, the WCET and synchronization estimations of phase II may be updated applying the results of phase IV. This leads to a closer estimation of the WCET and possibly some new variations of APDs are suitable for parallelization. Thus, a new Pareto front can be calculated. Afterwards, an APD with more realistic estimations can be chosen and implemented.

4 Tool Support

Jahr et al. neither describe any tools assisting the parallelization process, nor how the steps may be done efficiently. Therefore, we give a brief overview on some tools that support our extended parallelization approach in this section. The original Jahr-approach as well as our approach are model-based and therefore platform and language independent. However, in this section, we focus on tools for the programming language C. Other tools may be utilized for other programming languages, but also for C. Most of the following proposals were developed or extended during the parMERASA project [UBG$^+$13]. In all phases, standard UML tools may be used to handle the APD. Furthermore, the APD is additionally represented in XML to facilitate the modification of a particular model.

4.1 Tools for Phase I

Several tools were developed by Rapita Systems in their Rapita Verification Suite[5] (RVS) during parMERASA project, e.g. they provide a dependency tool. It utilizes a trace of a program execution to show all accesses to shared variables and the order of these accesses. Additionally, it supports a static mode to just make a list of all occurrences of shared variables.

A measurement-based WCET tool like RapiTime or a static timing analysis tool like OTAWA [BCRS11] has to be employed to determine the WCET of the different program segments.

Unfortunately, until now the code has to be analyzed manually to find situations where PDPs could be applied. We are currently discussing how this could work semi-automatically. A

[5]Homepage: `http://www.rapitasystems.com/` Licence: proprietary; tools for parallel analysis may not have been released yet. RVS includes e.g. RapiTime, RapiCheck, RapiTask and RapiCover.

first step could be the segmentation of code into different pieces based on the obtained data dependencies and existing control structures (conditional blocks, loops, etc.). The extracted segments, combined with the data dependencies, can be applied to suggest parallel sections.

Furthermore, tools for automatic parallelization may be utilized in an assisting role to find situations for parallel execution, see [KVP11, CMM10] for examples.

4.2 Tools for Phase II

To facilitate phase II, a speedup approximation and APD optimization tool has been developed, which is described in [JSK$^+$14]. It is open source and can be downloaded on GitHub[6]. The tool takes a XML file (representing an APD), a list of functions and all global variables accessed by the passed functions. During execution, it applies a genetic algorithm to find optimized APDs. Finally, we get the Pareto front (cf. section 2).

4.3 Tools for Phase III

At the implementation, we see potential to generate code (semi-)automatically: First, for the synchronization of shared variables, mutator methods (`get-/set-`functions) may be automatically generated including the necessary synchronization like e.g. locks or non-blocking functions (see [Her91, HLM03]).

Second, we developed a library composed of timing-analysable algorithmic skeletons (TAS) [SFJU15], which implement PDPs applicable for hard real-time systems [GJU13]. Thereby, these PDPs are the same as the ones usable in APDs. The library is open source[7], a detailed description how to use it can be found in [JSK$^+$14].

Combining the APD (in XML format) with our timing-analysable algorithmic skeletons (TAS), we have the idea that the skeletons could be automatically placed in code where PDPs are applied. Therefore, we plan to use the annotations of phase I, which tag the location of possible parallelisms. Applying the results of phase II, the PDPs may be switched on or off and a tool replaces the corresponding source code with code calling our library. Hence, the skeleton library can be seen as basis for a tool that provides support for implementing parallelization.

4.4 Tools for Phase IV

In this new phase, the functional and timing correctness of the parallel program is analysed. The static analysis tool OTAWA and the measurement-based tool RapiTime were extended

[6]Homepage: `https://www.github.com/parmerasa-uau/parallelism-optimization/` Licence: GNU LGPL v3

[7]Homepage: `https://www.github.com/parmerasa-uau/tas/` Licence: GNU LGPL v3

for the timing analysis of parallel code. These extensions are described in detail in [par14]. Here, we give a short overview on further extensions, which are described in the same deliverable: RapiCheck allows to check the functional equivalency between the parallel and sequential program and thus to verify if the parallelization was performed correctly. In case of a discovered functional failure, the trace viewing tool RapiTask may help to locate it and therefore to determine the step of the parallelization process where the failure occurs. Finally, RapiCover provides code coverage functionality to expose how thoroughly the code is tested. All these tools enable analysis of the resulting code of phase III and can be applied for measuring the quality of the implemented parallelization (i.e. speedups, holding the deadlines, constraint checking and code coverage).

The results determined with these tools show the real behaviour of the parallelized program on the parallel platform. Therefore, they form a good basis for an improved optimization in phase II.

5 Conclusion and Outlook

We presented an extension of the pattern-based parallelization approach which was originally introduced by Jahr et al. [JGU13b, JGU13a]. With the introduction of phase III, more attention is spent to the implementation. The additional phase for quality assurance checks the functional and timing correctness. Its results may be employed to iteratively improve the parallel program.

Although there is not yet a tool available to fully automate the parallelization process or the revealing of potential parallelism, we gave an overview over several assisting tools:
For **phase I**, there is a tool finding dependencies and creating a XML file of them. Further tools exist for timing analysis. We developed a tool for **phase II** which takes a highly parallel APD together with the WCETs of the program segments and a list of dependencies to find optimal APDs. At the implementation in **phase III**, we suggest using a code generator for the synchronization functions. Additionally, our algorithmic skeleton library eases the implementation of PDPs in the source code. We plan a tool to automate this process. In **phase IV**, a timing analysis of the parallel program is done which may lead to further improvements starting from phase II.

Altogether, we plan to further improve our tools and to develop some more tools to enable a (semi-)automated parallelization with our extended parallelization approach.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement no. 287519 (parMERASA). This project ended in september 2014.

References

[BCRS11] Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat. OTAWA: An Open Toolbox for Adaptive WCET Analysis. In *Software Technologies for Embedded and Ubiquitous Systems*, volume 6399 of *LNCS*, pages 35–46. Springer Berlin Heidelberg, 2011.

[CMM10] Daniel Cordes, Peter Marwedel, and Arindam Mallik. Automatic parallelization of embedded software using hierarchical task graphs and integer linear programming. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 267–276. IEEE, 2010.

[Fos95] Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman, Boston, MA, USA, 1995.

[GJU13] Mike Gerdes, Ralf Jahr, and Theo Ungerer. parMERASA Pattern Catalogue. Timing Predictable Parallel Design Patterns. Technical Report 2013-11, Department of Computer Science, University of Augsburg, Augsburg, Germany, 2013.

[Her91] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(1):124–149, 1991.

[HLM03] Maurice Herlihy, Victor Luchangco, and Mark Moir. Obstruction-free synchronization: Double-ended queues as an example. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 522–529. IEEE, 2003.

[JFG$^+$14] Ralf Jahr, Martin Frieb, Mike Gerdes, Theo Ungerer, Andreas Hugl, and Hans Regler. Paving the Way for Multi-cores in Industrial Hard Real-time Control Applications. In *WiP at 9th IEEE International Symposium on Industrial Embedded Systems (SIES)*, Pisa, Italy, 2014.

[JFGU14] Ralf Jahr, Martin Frieb, Mike Gerdes, and Theo Ungerer. Model-based Parallelization and Optimization of an Industrial Control Code. In *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme X, Schloss Dagstuhl, Germany, 2014, Tagungsband Modellbasierte Entwicklung eingebetteter Systeme*, pages 63–72, fortiss GmbH, Munich, March 2014.

[JGU13a] Ralf Jahr, Mike Gerdes, and Theo Ungerer. On Efficient and Effective Model-based Parallelization of Hard Real-Time Applications. In *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme IX*, pages 50–59, fortiss GmbH, Munich, April 2013.

[JGU13b] Ralf Jahr, Mike Gerdes, and Theo Ungerer. A pattern-supported parallelization approach. In *International Workshop on Programming Models and Applications for Multicores and Manycores*, PMAM '13, pages 53–62, New York, NY, USA, 2013. ACM.

[JGU$^+$14] Ralf Jahr, Mike Gerdes, Theo Ungerer, Haluk Ozaktas, Christine Rochange, and Pavel G. Zaykov. Effects of Structured Parallelism by Parallel Design Patterns on Embedded Hard Real-time Systems. In *20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Chongqing, China, August 2014.

[JSK$^+$14] Ralf Jahr, Alexander Stegmeier, Rolf Kiefhaber, Martin Frieb, and Theo Ungerer. User Manual for the Optimization and WCET Analysis of Software with Timing Analyzable Algorithmic Skeletons. Technical Report 2014-05, University of Augsburg, 2014.

[KVP11] Stefan Kempf, Ronald Veldema, and Michael Philippsen. Is There Hope for Automatic Parallelization of Legacy Industry Automation Applications? In GI, editor, *Proceedings of the 24th Workshop on Parallel Systems and Algorithms (PARS)*, pages 80–89, 2011.

[MBC11] Anne Meade, Jim Buckley, and John J. Collins. Challenges of Evolving Sequential to Parallel Code: An Exploratory Review. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL '11, pages 1–5, New York, NY, USA, 2011. ACM.

[MMS99] Berna L. Massingill, Timothy G. Mattson, and Beverly A. Sanders. Patterns for Parallel Application Programs. *Proceedings of the 6th Conference on Pattern Languages of Programs (PLoP'99)*, 1999.

[MSM04] Timothy G. Mattson, Beverly A. Sanders, and Berna L. Massingill. *Patterns for parallel programming*. Addison-Wesley Professional, first edition, 2004.

[par14] Report on support of tools for case studies. Deliverable 3.12 of the parMERASA project, September 2014. `http://www.parmerasa.eu/files/deliverables/Deliverable_3_12.pdf`.

[PQnZ+14] Miloš Panić, Eduardo Quiñones, Pavel G. Zaykov, Carles Hernandez, Jaume Abella, and Francisco J. Cazorla. Parallel Many-core Avionics Systems. In *14th International Conference on Embedded Software*, EMSOFT '14, pages 26:1–26:10, New York, NY, USA, 2014. ACM.

[SFJU15] A. Stegmeier, M. Frieb, R. Jahr, and T. Ungerer. Algorithmic Skeletons for Parallelization of Embedded Real-time Systems. In *3rd Workshop on High-performance and Real-time Embedded Systems (HiRES)*, 2015.

[UBG+13] T. Ungerer, C. Bradatsch, M. Gerdes, F. Kluge, R. Jahr, J. Mische, J. Fernandes, P.G. Zaykov, Z. Petrov, B. Boddeker, S. Kehr, H. Regler, A. Hugl, C. Rochange, H. Ozaktas, H. Casse, A. Bonenfant, P. Sainrat, I. Broster, N. Lay, D. George, E. Quinones, M. Panic, J. Abella, F. Cazorla, S. Uhrig, M. Rohde, and A. Pyka. parMERASA – Multi-core Execution of Parallelised Hard Real-Time Applications Supporting Analysability. In *2013 Euromicro Conference on Digital System Design (DSD)*, pages 363–370, Sept 2013.

[UBG+15] T. Ungerer, C. Bradatsch, M. Gerdes, F. Kluge, R. Jahr, J. Mische, A. Stegmeier, M. Frieb, J. Fernandes, P.G. Zaykov, Z. Petrov, B. Boddeker, S. Kehr, H. Regler, A. Hugl, C. Rochange, H. Ozaktas, H. Casse, A. Bonenfant, P. Sainrat, I. Broster, N. Lay, D. George, E. Quinones, M. Panic, J. Abella, F. Cazorla, S. Uhrig, M. Rohde, and A. Pyka. Experiences and Results of Parallelisation of Industrial Hard Real-time Applications for the parMERASA Multi-core. In *3rd Workshop on High-performance and Real-time Embedded Systems (HiRES)*, 2015.

[WEE+08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The Worst-case Execution-time Problem – Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36:1–36:53, May 2008.