Communication Patterns for Embedded Real-Time Many-Cores

Alexander Stegmeier*,1, Jörg Mische*,1,
Theo Ungerer*,1

* *Department of Computer Science, University of Augsburg, Universitaetsstrasse 6a, 86159 Augsburg, Germany*

ABSTRACT

We envision that multi- and many-core processors will be applied in future embedded real-time systems in automotive, avionics and automation domains. Embedded multi-cores feature a shared memory, whereas many-cores tend towards distributed memory possibly with pure message-passing. We further envision the parallelization of real-time applications. Thus, we plan to provide assistance for parallel programming by delivering a higher level of abstraction to the programmer. The abstraction is introduced by parallel communication patterns that are timing predictable, ease WCET analysis, simplify the programming of parallel code and are applicable for shared-memory multi-cores and message-passing many-cores. In this paper, we briefly describe the approach of our communication patterns and emerging research topics.

KEYWORDS: parallel programming; real-time; many-core

1 Introduction

More and more embedded systems tend to utilize multi- and many-cores as hardware platform. Due to the rapid changes in hardware platforms, it is not possible to predict a particular kind of platform as standard for the long term. Hence, there is a necessity for platform independent parallel implementation libraries for embedded systems.

A major part in the area of embedded systems is covered by real-time applications. This kind of application requires a worst-case execution time (WCET) estimation and thus, must be written in an timing analysable way. Typically, tools for WCET analysis such as OTAWA [BCRS11] process on object code, but require additional input settings only available in source code. WCET analysis in parallel code faces even more obstacles. In parallel code the informations implicitly provided in code are often not sufficient to gain an acceptable estimation of the WCET or even to enable an estimation at all.

Furthermore, programming libraries must be practical and easy to use. As the parallel execution of programs is not intuitive, the programming of parallel code can be simplified by an abstraction from parallel programming details. Such abstractions are researched by

[1]E-mail: {alexander.stegmeier,mische,ungerer}@informatik.uni-augsburg.de

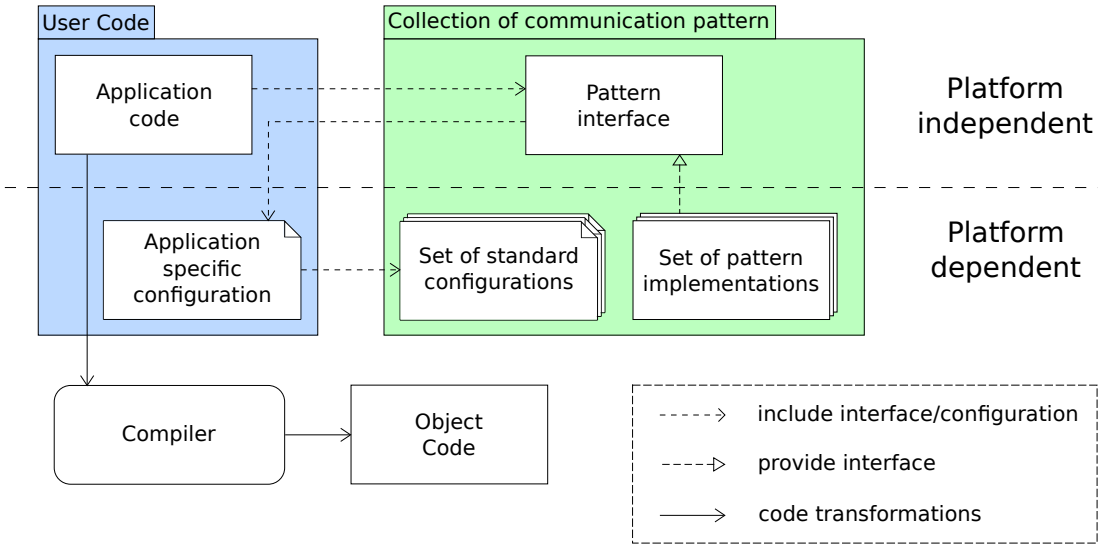

Figure 1: Overview of the of usage of the communication patterns

parallel design patterns [MSM04] and have been further developed to timing predictable parallel design patterns [GJU13] in parMERASA project [UBG+13].

Our research target is to provide abstract communication patterns to the programmer. In doing so, our objectives are to provide a platform independent pattern interface which can be ported and optimized to a particular platform with a minimal programming overhead. Furthermore, we plan to provide additional information for the WCET analysis by composing basic communication patterns to more abstract patterns which implicitly include more information about their behavior. Since these abstractions of patterns are masking details of parallel programming, they are particularly targeting the need of the programmer.

The remainder is structured as follows. Section 2 describes the approach of our communication patterns and section 3 states the major research topics at developing these patterns.

2 Approach

It is necessary to separate the application code in distinct execution and communication phases. This requirement enables a separated WCET analysis of sequential calculations and parallel communication and thus, yield to an improved WCET estimation. Furthermore, the communication can be distinguished in synchronization and data exchange. Both kinds of communications can be applied for different sets of execution threads. The pattern collection primarily concentrates on implementation of the communication phases.

2.1 Collection Overview

Figure 1 illustrates the usage of the communication patterns. In order to apply our collection of communication patterns, the programmer has to prepare its application code and additionally, a configuration file. Thereby, the configuration file (application specific configurations) includes information about the desired target platform.

The collection contains a pattern interface that provides access to the communication patterns and can be included and used in the application code. Furthermore, there are dif-

ferent implementations for each communication pattern and a set of standard configurations which can be included in the programmer's configuration file to reduce the configuration effort for the target platform. The utilized implementation for a pattern varies depending on the platform configured in the configuration file. Therefore, the application specific configurations are included in the pattern interface. At compile time the configurations are interpreted and based thereon one particular implementation for a pattern is chosen.

So far, the configurations mainly describe the chosen memory model, which can be a distributed or a shared memory model. Regarding to the memory models, two different implementations are provided thus far: a POSIX-based implementation for shared memory and a message-based implementation for distributed memory.

2.2 Provided Abstractions

The pattern collection will mainly provide two kinds of abstractions. These are a) several levels of transparency in the sense of hiding implementation details e.g. lower level communication details and b) several levels of compositions of lower-leveled communication patterns. Thereby, it will be taken care about making all levels of abstraction available to the user. This ensures the facilitation of parallel programming and concurrently enables the implementation of fine-grained communication optimizations.

Up to now, we see several parallel programming details, which can be masked by a transparency abstraction. An example for such details could be the mapping of global data to particular (distributed) memories or the mapping of threads to a core. Thereby, it is conceivable that the order of transparency levels is not strict. Hence, it could be possible that one pattern provides transparency about data distribution but threads have to be mapped explicitly, while another pattern provides the transparencies vice versa.

Another way of abstraction is the composition of lower-leveled communication pattern to high-level pattern. Therefore, a communication pattern utilizes several other patterns to provide a communication pattern of a higher level. Furthermore, when applying compositions recursively, multiple levels of abstraction can be built.

An example for this kind of abstraction could be a pattern *global sum* which sums up values of several cores and provides the result to all participants. In a message-passing model this pattern can be constructed with *reduction with summation* and *broadcast* which are in turn formed with point-to-points communications. For the POSIX model, *global sum* is composed of *Exclusive Add* and *Synchronous Read*. These are in turn composed of *Mutex* and *Barrier*.

This example shows that the advantage of abstract patterns is that they are implicitly including many information about the behavior of the individual cores. Thus, the behavior of a core at a specific time can be limited. Considering each low-level pattern by its own they obtain relatively high WCETs and a combination of them raises the over-estimation even more. However, if the exact sense of a communication is known, than it is possible to exclude hypothetical cases. Furthermore, the example demonstrates how abstractions can provide a common communication interface for parallel programs targeting different platforms.

3 Research Topics

One challenge is the identification of relevant communication patterns. An analysis of existing parallel applications that are relevant in embedded real-time systems and a comparison

with parallel design patterns [MSM04] could obtain commonly used patterns.

After identification of a pattern, it is necessary to implement it for all kinds of platforms provided by the collection. A couple of research questions arise here. Some of them deal with the problem of transforming a common interface to implementations for different platforms while the interface remains intuitive to the user. Furthermore, the implementation should keep the advantages provided by the programming models for the different platforms. We plan to provide assistance for shared memory systems (POSIX) and for systems utilizing distributed memory (message-passing). Moreover, it is planed to port the library to an embedded real-time many-core which is currently under development at University of Augsburg [MMU14].

A further topic is the classification of identified communication patterns. This includes the determination of characteristics of the individual pattern and the relationships to other patterns. Hence, the pattern collection targets real-time applications, a major individual characteristic will be the behavior at a WCET analysis. Further features could be the behavior of the average-case execution time (ACET) and provided transparencies. The relationship to other patterns aims at a statement about how they are composed and at a classification of the pattern in groups obtaining similar characteristics.

It is to mention, that the collection of communication patterns and its components i.e. *pattern interface*, the *set of implementations* and the *standard configurations* are still under development respectively future work.

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.

[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.

[MMU14] Jörg Mische, Stefan Metzlaff, and Theo Ungerer. Distributed memory on chip-bringing together low power and real-time. In *Proceedings of the Workshop on Reconciling Performance and Predictability (RePP)*, 2014.

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

[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.