Programming Heterogeneous MPSoCs using MAPS

Weihua Sheng, Jeronimo Castrillon, Anastasia Stulova, Maximilian Odendahl, Rainer Leupers, Gerd Ascheid
Institute for Communication Technologies and Embedded Systems
RWTH Aachen University, Germany
sheng@ice.rwth-aachen.de

ABSTRACT
Programming heterogeneous MPSoCs (Multi-Processor Systems on Chip) is a grand challenge for SoC providers and users today. The HW integration level of commercial MPSoC platforms increases at a fast pace. However, lacking tools to support MPSoC programming results in a much slower pace of SW productivity, which hampers deploying MPSoCs significantly. In this paper, we introduce the MAPS (MPSoC Application Programming Studio), a programming tool suite for heterogeneous MPSoC architectures. It uses both sequential C and a C language extension for describing applications in the form of process networks, and it performs optimized temporal and spatial task-to-processor mapping for MPSoC platforms. Several case studies have been presented to show the capabilities of MAPS as a promising solution for MPSoC compilation and programming.

1. PROBLEM STATEMENT
Heterogeneous MPSoCs are widely used in modern embedded systems. Programming such platforms remains a grand challenge for SoC providers and users today [1]. What is the right MPSoC programming model that captures both parallel computations and certified sequential C code? How to achieve optimized temporal and spatial task-to-processor mapping to meet real-time constraints? How to parallelize legacy C code? How to explore the vast SW mapping design space? Those are just a few examples of the multitude of SW design issues that MPSoCs pose.

To illustrate the problem of MPSoC programming, Fig. 1 shows a comparison of programming flows for uni-processor systems and for MPSoCs. In the uni-processor flow, SW programmers follow the sequential programming model (C being the most popular language) and rely on the compilers to generate target-specific code correctly and optimally, as shown in Fig. 1 (a). This has been a successful practice for the past few decades before MPSoCs due to the heroic role of compilers. However, traditional compiler technology does not scale for MPSoC. MPSoC SW requirements are doubled every ten months while SW productivity is doubled only every 2 years [2]. Fig. 1 (b) demonstrates the current problematic programming flow for MPSoCs, which results in low SW productivity. Applications firstly need to be partitioned in parallel, followed by the step of spatial and temporal mapping of those partitioned tasks onto MPSoC processing elements. Heterogeneous MPSoC programmable processing elements often come with their own SW stack (API, OS) and own compiler. Therefore, after partitioning and mapping, correct code must be generated respectively for those processing elements to be further compiled. This process is currently manual and error-prone, let alone more iterations are usually needed to reach the optimized performance.

Compared to the uni-processor, the programming flow for MPSoCs obviously involves now more tasks which fall on the programmers’ shoulders, e.g. partitioning and mapping. Those are non-trivial tasks. As the counterpart of traditional compiler in the uni-processor programming, little tools support exists in the MPSoC programming.

2. MAPS OVERVIEW
As a result of a long-term R&D investment, MAPS Compiler has been developed at ICE of RWTH Aachen University, which is a tool framework with an Eclipse-based IDE that eases programming of heterogeneous MPSoC architectures. The main features of MAPS are:

- Compilation framework for multi-core systems: MAPS compiler uses both sequential C and a C language extension for describing applications in the form of process networks as inputs. It performs source-to-source translation to generate target specific code to leverage the existing C compiler technology for multi-core processing elements. By executing the code on a real or virtual target platform, the user can quickly evaluate the result quality and, if required, explore further SW mapping options. The compiler framework is retargetable thus minimizing the SW tools investment as well.

Figure 1: Programming Flow (a) Uni-processor (b) MPSoC
The inputs to the MAPS compiler are the programs written. We have retargeted MAPS compiler towards OMAP3530 [5] multi-core programmers. All those add to a high entrance barrier for communications. Compiler tool chains are also separate on processors. TI proprietary SW runs on the DSP and provides a DSP/Link layer which handles the inter-processor communication. Linux OS running ARM is maintained by open-source community. TI proprietary SW runs on the DSP and provides a DSP/Bios [8] from TI, Linux is used as OS, while on the DSP side a lightweight multi-tasking operating system, called DSP/Bios [8] from TI, is used. While OMAP exhibits a typical heterogeneous HW set-up, its software tool chains are also heterogeneous. For instance, the Linux OS running ARM is maintained by open-source community. TI proprietary SW runs on the DSP and provides a DSP/Link layer which handles the inter-processor communications. Compiler tool chains are also separate on the processors. All those add to a high entrance barrier for multi-core programmers.

We have retargeted MAPS compiler towards OMAP3530 [5]. The inputs to the MAPS compiler are the programs written.
technology in which waveforms are composed out of library blocks called Nuclei, which are algorithmic entities that represent demanding computational kernels that are common across different wireless communication standards. A Nucleus can be implemented in different platforms in several ways. Each of these implementations is called a flavor, including platform dependent implementation details, several of which are crucial for the tool flow. In the Nucleus Concept, a flow was envisioned in which a mapper selects the best matching flavors to implement a waveform under time and energy constraints (see Figure 3).

The Nucleus tool flow – an implementation based on the Nucleus Concept is presented in [12]. The flow is built on top of the MAPS framework and integrated with state-of-the-art tools for system level design. As a Waveform Description Language (WDL), the CPN language of the MAPS framework is used. This language allows to specify applications following the KPN model. The Nucleus Mapper selects the best flavors from a given platform to implement the waveform. The best options are exported in form of simulation models at different levels of abstraction. Enabled by the MAPS compiler, for functional verification and debugging, abstract level simulators based on the Virtual Processing Unit (VPU) technology [13] are automatically generated. Low level simulators for timing verification are set up in board support packages (Platform description, flavor library).

The Nucleus tool flow is used. This language allows to specify applications following the KPN model. The Nucleus Mapper selects the best flavors from a given platform to implement the waveform. The best options are exported in form of simulation models at different levels of abstraction. Enabled by the MAPS compiler, for functional verification and debugging, abstract level simulators based on the Virtual Processing Unit (VPU) technology [13] are automatically generated. Low level simulators for timing verification are set up in board support packages (Platform description, flavor library).

4. SUMMARY
This paper presents the MAPS compiler which aims at tackling the challenge of MPSoC programming. It is a tool framework with an Eclipse-based IDE that eases programming of heterogeneous MPSoC architectures, while ensuring optimized system performance. Over several years of R&D, MAPS has taken the step from basic research to a tool framework that enables programming of real-life complex MPSoC platforms. We plan to present this new technology to a wider audience, receive feedback for future enhancements, and make MAPS accessible to early adopters in industry.

5. ACKNOWLEDGMENT
This work has been supported by the UMIC Research Centre, RWTH Aachen University.

6. REFERENCES