Towards Effective Parallelization and Accelerator Offloading for Heterogeneous Multicore Embedded Systems

Miguel Angel Aguilar*,
Rainer Leupers*

* *Institute for Communication Technologies and Embedded Systems, RWTH Aachen University, Germany*

ABSTRACT

The use of MPSoCs is a common practice in the design of state-of-the-art embedded devices. However, MPSoC programming is still a challenging task that currently involves several manual steps. This task becomes even more challenging for heterogeneous architectures, where general purpose processors are combined with accelerators. In this work, we describe an approach that jointly addresses extraction of parallelism and accelerator offloading from sequential applications targeting heterogeneous MPSoCs. Results show the effectiveness of our approach, as we are able to speedup sequential benchmarks significantly on a commercial platform.

KEYWORDS: Parallelization; Offloading; Embedded Software; Heterogeneous MPSoCs

1 Introduction

MPSoCs have emerged as a response to the demands of applications in the embedded industry such as performance, power and cost. The current design trend of state-of-the-art MPSoCs is towards heterogeneous architectures, where cores with different attributes, such as Instruction Set Architectures (ISAs) and clock frequencies are combined. Typically these platforms integrate general purpose processors, such as ARMs, with special purpose cores, such as DSPs or GPUs. In order to exploit the full potential of an MPSoC, applications have to be optimized for these platforms. This is a challenging task that has been addressed in two complementary directions: *i*) *via new parallel programming paradigms*, and *ii*) *using automatic parallelization frameworks*.

Several programming paradigms have been proposed to express parallelism, such as the OpenMP industry standard. In addition, in the embedded domain dataflow Models of Computation (MoCs), such as Kahn Process Networks (KPNs) [Gil74] have gained acceptance. *C for Process Networks* (CPN) [SLX] is an example of a commercial solution that allows to implement KPNs.

[1]E-mail: {aguilar, leupers}ice.rwth-aachen.de

Despite the efforts for providing a convenient parallel programming paradigm, the programmer still has the cumbersome task of writing correct parallel code. In practice, this task is even more challenging considering that in general software development is about evolution and transformation [Joh10], where developers have to optimize legacy sequential code for MPSoCs. This is an extremely error-prone and time consuming process that involve several manual steps: identifying computational intensive code sections, identifying data dependencies, extracting profitable parallelism opportunities, deciding in which type of core to map code sections in heterogeneous platforms, and finally writing correct parallel code.

The most prominent solution is to automate these steps by using parallelization frameworks. Therefore, our research is focused on developing a joint parallelization and offloading method to optimize legacy sequential C applications for heterogeneous embedded MPSoCs [AER+15, ALAM16]. Our ultimate goal is to provide an effective toolflow that fulfills the needs of a realistic industrial environment in the embedded domain. The main contributions of our work are summarized as follows:

- A framework for parallelization and offloading of code regions in sequential applications targeting heterogeneous multicore embedded devices.
- Realization of the optimization opportunities by using OpenMP and CPN.
- Evaluation of the approach by optimizing multiple realistic benchmarks on a heterogeneous MPSoC from Texas Instruments.

2 Background and Related Work

Research efforts on automatic parallelization have left valuable techniques and observations. However, there is not yet a widely accepted solution and many issues remain open, especially in the embedded domain. Early works focused on extraction of *data level parallelism* from loops with no carried dependencies. While this form of parallelism is abundant in scientific applications, works such as [Isl07] show that in the embedded domain other parallelism patterns should be explored. In addition, current parallelizing compilers rely only on *static information* obtained at compile time. However, this approach often fails in languages like C as it is extremely conservative with respect to the analysis of pointers, dynamic allocated memory and indirect function calls [Tou11]. Therefore, the use of *dynamic analysis* to obtain runtime information is proposed as an alternative or a complement to static analysis [Tou11]. Moreover, existing automatic parallelization frameworks for embedded MPSoCs either do not consider heterogeneous platforms at all [CL14], or are only capable of targeting platforms with cores of the same ISA that run at different frequencies [Cor13].

3 Proposed Approach

Figure 1 shows the main components of the proposed toolflow. It starts with a sequential C application, a model of the MPSoC and constraints provided by the developer to guide the analysis. Then a *program model* that contains both performance and control/data dependency information is built in a hybrid fashion by combining *static* (at compile time) and *dynamic* (at runtime) analyses. The program model is analyzed by algorithms that focus on computationally intensive code regions to identify multiple parallel patterns, and to decide which regions are good candidates for accelerator offloading. In first place the resulting

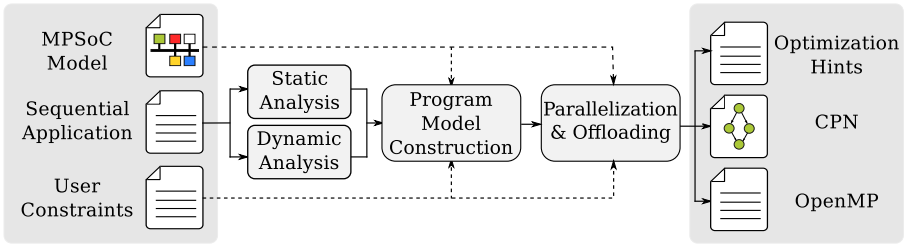

Figure 1: Parallelization and Offloading Toolflow

information is used to generate source level hints that provide developers with a deep understanding of parallelization and offloading opportunities. Moreover, the resulting information also enables code generation in a parallel paradigm, such as OpenMP or CPN.

4 Results

In order to evaluate the effectiveness of our approach we selected the 66AK2H MPSoC [Ins], which belongs to the Keystone II family from Texas Instruments. This MPSoC is composed of a cluster with four ARM Cortex-A15 cores and a cluster with eight C66x DSP cores. The goal of this evaluation is to parallelize and offload loop regions on the Keystone II platform using OpenMP. For this purpose we selected the Rodinia benchmark suite, which targets heterogeneous platforms. Rodinia is a set of data parallel benchmarks manually annotated with OpenMP pragmas and other paradigms as well. This is beneficial to verify the effectiveness of our approach, since it allows to compare the automatically generated annotations by our toolflow against the manual annotations by expert programmers.

Table 1 shows the number of manually versus automatically parallelized and offloaded loops with OpenMP. From these results and by comparing the generated code versus the handwritten code, we could verify that our framework was able to parallelize and offload the same loops in almost all benchmarks, just with two exceptions. This first one was in the *BP* case in which we offloaded one loop in contrast to the manual version. As we will see in the speedup results this leads to a performance improvement. The other exception was in the *ParticleFilter* case in which we annotated only two loops, in contrast to eleven loops in the manual version. The reason for this difference is that in the manual version of *ParticleFilter* several non-intensive loops were annotated.

Figure 2 shows the estimated speedup results obtained after parallelize and offload loops within the Rodinia benchmarks with our framework. The baseline for speedup computation is the sequential execution time of each benchmark on a single ARM core. We evaluated two scenarios: *i*) only parallelization on the ARM cluster, and *ii*) parallelization and offloading considering both ARM and DSP clusters. In the first scenario the performance was increased by 3.3× in the best case, and on average by 2.3×. In the second scenario, results show that the performance improvements for benchmarks with offloaded loops outperform the first scenario (*BP*, *BFS*, *Hotspot* and *LUD*). These results highlight the benefit of our joint parallelization and offloading approach. Here the performance of the benchmarks is increased in the best case by 5×, and on average by 2.8×.

Table 1: Loop Parallelization and Offloading with OpenMP

Benchmark	Manual		Automatic	
	Parallelized	Offloaded	Parallelized	Offloaded
BP	2	0	2	1
BFS	2	1	2	1
Hotspot	1	1	1	1
Kmeans	1	0	1	0
LUD	2	1	2	1
NN	1	0	1	0
ParticleFilter	11	0	2	0
PathFinder	1	0	1	0

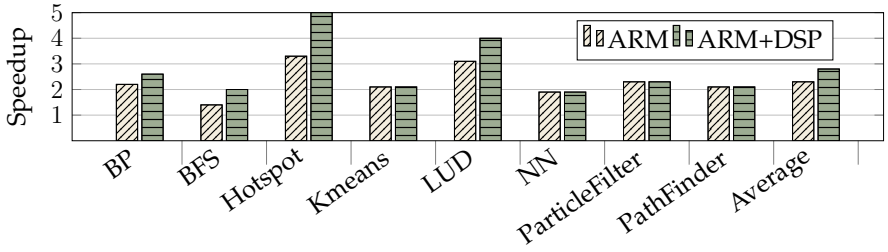

Figure 2: Estimated Speedups on Keystone II Platform

References

[AER+15] Miguel Angel Aguilar, Juan Fernando Eusse, Projjol Ray, Rainer Leupers, Gerd Ascheid, Weihua Sheng, and Prashant Sharma. Parallelism extraction in embedded software for Android devices. In *Proc. SAMOS XV*, July 2015.

[ALAM16] Miguel Angel Aguilar, Rainer Leupers, Gerd Ascheid, and Luis Gabriel Murillo. Parallelism extraction in embedded software for Android devices. In *Proc. 53rd DAC*, June 2016.

[CL14] Jeronimo Castrillon and Rainer Leupers. *Tool Flows to Close the Software Productivity Gap*. Springer, 2014.

[Cor13] Daniel A. Cordes. *Automatic Parallelization for Embedded Multi-Core Systems using High-Level Cost Models*. PhD thesis, TU Dortmund University, 2013.

[Gil74] Gilles Kahn. The Semantics of a Simple Language for Parallel Programming. In *IFIP Congress 74*, pages 471–475, North Holland, Amsterdam, 1974.

[Ins] Texas Instruments. 66AK2H14/12/06 Multicore DSP+ARM Keystone II SoC. SPRS866.

[Isl07] M.M. Islam. On the limitations of compilers to exploit thread-level parallelism in embedded applications. In *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on*, pages 60–66, July 2007.

[Joh10] Ralph E. Johnson. Software development is program transformation. In *Proc. FoSER*, pages 177–180, 2010.

[SLX] Silexica. [Online] Available `http://www.silexica.com` (accessed 5/2016).

[Tou11] Georgios Tournavitis. *Profile-driven Parallelization of Sequential Programs*. PhD thesis, University of Edinburgh, 2011.