Introduction

Performance increase, in terms of faster execution and higher energy efficiency, is a never-ending research domain and does not come for free. Living in an era where there is an immense amount of data, the demand for data processing by modern computing systems rises even more. Technological giants, such as Google and Facebook gather and compute a lot of data, for instance during Machine Learning related applications and lengthy simulations. This large amount of data processing requires a lot of computational power and ends up in lengthier and lengthier execution latency time.

The breakdown of Dennard scaling [2], along with the seemingly inevitable end of Moore's law economic aspect [5], present a new challenge to computer architects striving to achieve better performance in the modern computer systems. Heterogeneous computing is emerging as one of the solutions in order to keep the performance trend rising. This is achieved by turning the focus to specialized Hardware (HW) that can accelerate the execution of a Software (SW) application or a part of that application.

Instead of a general purpose CPU, or parallel CPUs, managing the execution of SW applications, specialized pieces of HW, namely accelerators, can be used alongside with a general purpose CPU and execute the most demanding parts of an application, in terms of computation. Consequently, the need for a powerful single CPU is no more that critical, as the execution can be offloaded to other pieces of HW as well. The gain is twofold, as we both achieve a more balanced execution with the use of different HW resources, and we offload the execution of specific, more demanding parts of the computation to specialized HW accelerators.

One example of a widely spread heterogeneous architecture is the addition of a GPU to a CPU on the same chip, in order to exploit the parallelism and computing power that a GPU has to offer, when it comes to image processing and 3D graphics rendering. Other examples are general purpose CPUs, coupled with dedicated HW that execute specific kernels or even full applications. The latter architecture could come in a number of variations, with one or more HW accelerators, and different types of coupling, tightly or loosely [1]. The design of the first option, tightly or co-processor model, is done by using the accelerator as an Instruction Set Extension in the default pipeline of the CPU. The latter implements the connection between CPU and accelerator loosely, without any knowledge of the underlying CPU micro-architecture.

The goal is to design efficient HW/SW computer architectures, so that the time latency and energy requirements are ever decreasing. The heterogeneous system that I considered during my research comprises a general purpose CPU, loosely coupled with a number of specialized HW accelerators, dedicated to the acceleration of specific parts of an application.

The choice of which parts of an application to be accelerated, though, as well as the type of accelerators to be used, while taking into account the underlying memory system, are all non-trivial research questions and depend heavily on the SW applications characteristics that are going to be accelerated. In addition to the accelerator selection problem, every HW accelerator can be designed with a number of optimizations embedded onto it, according to the execution task characteristics that is targeted for acceleration. For instance, in the case that a loop is included in the execution, there could be a loop unrolling factor taken into account during the synthesis of the accelerator that may dramatically affect the outcome. Another example would be the addition of a memory buffer, e.g. a scratchpad memory, to reduce the memory latency of the execution. Furthermore, the underlying memory system, as in every computer architecture, can significantly affect the overall performance, due to communication latency, and should be taken into account during the selection of the accelerators to be implemented, along with their respective potential optimizations.

Therefore, an in-depth SW analysis can be crucial, prior to designing a heterogeneous system, as it can provide valuable information and subsequently highly benefit performance. My research during my PhD has revolved around various ways that SW analysis, by extending the LLVM compiler framework [3] and, hence, its potential, can guide the engineer synthesizing HW accelerators by making informed decisions early in the development cycle.

An overview of the research conducted during my PhD is depicted in Figure 1. This can be viewed as a map of the current PhD thesis in order to navigate throughout my research time-line and present a high level view of how each piece is connected to each other.

Chapter 1 answers the question of *what* should be accelerated, namely which parts of computation, given a constraint on HW area resources. Under the scope of this Chapter the RegionSeeker tool-chain is presented. RegionSeeker is an LLVM based framework that, given a SW application provided as input, identifies

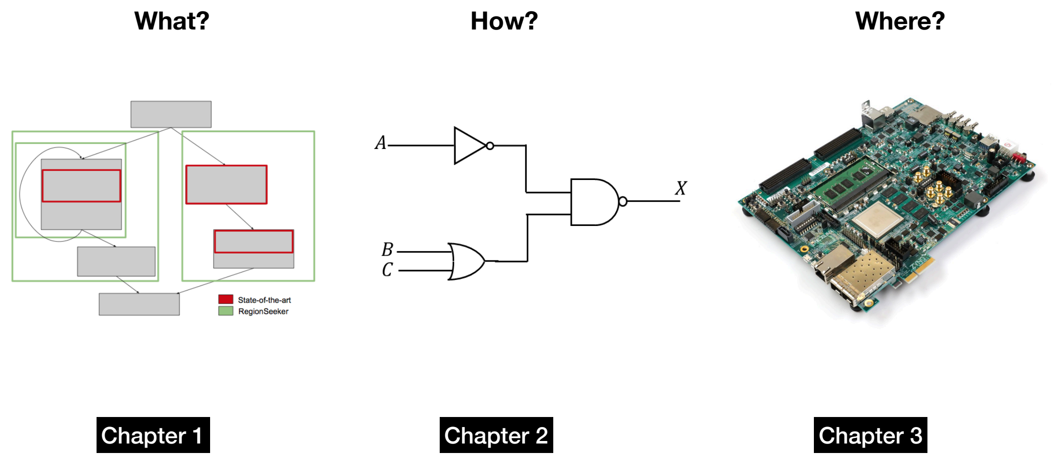

Figure 1. Overview of the research that has been conducted during my PhD and the respective Chapters of the PhD thesis.

and selects, in a fully automatic fashion, HW accelerators under the constraint of an area (HW resources) budget. The granularity of the candidates for acceleration considered is that of a subgraph of the control flow graph of a function, with a single control input and a single control output. These candidates are called regions. After identification takes place, a selection algorithm solves the problem optimally for finding the subset of the initial regions list that, under a given area constraint, maximizes the collective speedup obtained. The evaluation of RegionSeeker took place by using both an industrial tool, such as Xilinx Vivado HLS [6], and a research HW accelerator simulator, such as Aladdin [4].

In Chapter 2, the analysis that is presented attempts to answer the research question of *how* the identified and selected HW accelerators should be implemented in order to achieve improved performance. Under that scope, Data Reuse analysis, during the execution of a specific domain of applications, reveals the effectiveness of private local memory structures. Furthermore, for HW accelerators that contain loops, an optimal Loop Unrolling factor can be predicted for each of the included loops. The optimal Loop Unrolling factor for each loop is defined according to the target of optimization, which can be either less use of HW resources or better speedup. With the aid of a prior LLVM based analysis of the loops and Machine Learning classification, predictions can be performed on a set of loops and the respective Loop Unrolling factors may be subsequently applied during the synthesis phase of the accelerators.

Finally, Chapter 3 tackles the research question of what should be accelerated but at the same time taking into account *where* the specialized HW is hosted. An analysis of the system at hand and its memory hierarchy can affect vastly the selection of HW accelerators and subsequently the performance achieved. Latency due to data exchange between the HW accelerators and the main memory could add a significant overhead to the overall computation time. In this Chapter the AccelSeeker, an LLVM based tool-chain, is presented. AccelSeeker performs thorough analysis of applications and estimates memory latency along with computational latency of candidates for acceleration. The granularity of the candidates is that of a subgraph of the entire call graph of the application, with a root function (node) and zero outgoing edges of the identified subgraph. HW accelerators are selected by an algorithm so that speedup, or energy efficiency, is maximized, under a given area budget. The evaluation of AccelSeeker took place on Zynq UltraScale platform by Xilinx, considering a demanding and complex application such as H.264.

Bibliography

[1] E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni. An analysis of accelerator coupling in heterogeneous architectures. In *Proceedings of the 52nd Design Automation Conference*, pages 1–6. ACM, June 2015.

[2] H. Esmaeilzadeh, E. Blem, R. St Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 365–376, 2011.

[3] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the 2nd International Symposium on Code Generation and Optimization*, pages 75–88, Mar. 2004.

[4] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks. Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *Proceedings of the 41st Annual International Symposium on Computer Architecture*, pages 97–108. IEEE, July 2014.

[5] T. Simonite. Moore's law is dead. Now what? *MIT Technology Review, May*, 13:40–41, 2016.

[6] Xilinx. Vivado high-level synthesis. `www.xilinx.com/products/design-tools/vivado/integration/esl-design.html`, Mar. 2017.