Chapter 6
Adaptive Computing Systems and Their Design Tools

Andreas Koch

Abstract While reconfigurable adaptive computing has many proven advantages over conventional processors, in practice, it is often limited to niche applications. This situation, which we aim to resolve with our research, is often linked to the lack of programming languages for adaptive computers that are familiar to software developers. We present a compile flow capable of translating general-purpose C programs to hybrid hardware/software applications for execution on an adaptive computer and give an overview of the required advances in compiler technology as well as in computer architecture and operating system design.

6.1 Introduction

As demonstrated numerous times, reconfigurable computing can have significant advantages over conventional processors for a wide range of applications [28]. Despite these advantages, however, it is only rarely employed outside of academic settings.

One of the key reasons for this discrepancy is the difficulty of actually programming a reconfigurable computer. Most commonly, this is done by designing a compute architecture for the algorithm from scratch, which is then described in a hardware design language such as VHDL or Verilog.

While this approach can result in very high-performance implementations, it requires programmers to be experienced in computer architecture, digital logic design and hardware design languages. Only very few software developers actually have these skills. Thus, the power of reconfigurable computing remains unavailable to most potential users.

Andreas Koch
Embedded Systems and Applications Group, Dept. of Computer Science, Technische Universität Darmstadt, Darmstadt Germany, e-mail: koch@esa.cs.tu-darmstadt.de
In recent years, many attempts have been made to close this gap and lift the abstraction level of reconfigurable computer programming to that of conventional software-programmable processors.

To this end, we have been working on Comrade, a compiler for automatically translating general-purpose ANSI C programs for computers containing both a conventional and a reconfigurable processor. As we will describe below, the choice of C with its pointers and possibly irregular control flow has significant effects on both the compile flow as well as the target computer architecture and its operating system.

Our DFG-funded project “Adaptive Computing Systems and their Design Tools” was initiated prior to the DFG Priority Program 1148, but has been associated with the Program right from the start due to the thematic closeness. Since the schedules of our on-going project and the concluding Priority Program are thus out-of-phase, this report will concentrate on the major results achieved during the era of the SPP. Section 6.7 will give some perspective on the issues we are addressing in our current research.

6.2 Execution Model

In contrast to traditional research on High-Level Synthesis (HLS) [8] our target architecture is assumed to always contain a conventional software-programmable processor (SPP) in addition to a reconfigurable processor. While the compute-intensive parts of a program can be implemented in a spatially distributed fashion for high-performance, other parts of the program that are either unsuitable (e.g., I/O using printf or similar functions) or that are only used rarely and would not justify the permanent allocation of computing area (e.g., error handling) are left in software on the SPP.

This target architecture also avoids a basic problem of High-Level Synthesis, which aimed to translate the entire program into hardware: If the input program contained a construct (e.g., function calls, irregular control flow, dynamic memory allocation, etc.) that the specific HLS algorithm could not handle, the translation was aborted completely. While we also intend to translate our input language to the widest practical degree, our flow can always fall back to the SPP to execute program parts that the flow cannot process yet due to implementation limitations, or that would exceed the capacity of the reconfigurable device. This allows incremental development of the compiler, with increasing parts of the profitable computations of a program being moved for acceleration to the reconfigurable device.

We call such an architecture an adaptive computer system (ACS). To be more precise, we differentiate between the underlying reconfigurable device (RD), which can be an either an FPGA or a coarse-grained reconfigurable array (CGRA), and the reconfigurable compute unit(s) (RCU) that can be mapped to it.

When combining multiple processing elements (such as the SPP and RCU of an ACS), the manner of their interaction must be specified. This is done by the execution model (discussed in greater detail in [22]).