CLooGVHDL and JCCI

Harald Devos, Wim Meeus, Dirk Stroobandt

Harald.Devos@UGent.be

ELIS-PARIS – Ghent University – Belgium

http://www.elis.ugent.be/

Abstract

CLooGVHDL and JCCI offer an extendible C-to-VHDL framework to develop high-level synthesis techniques for data-intensive applications on heterogeneous memory systems.

1. Introduction

Multimedia applications are an example of applications that are not only computation-intensive but also data-intensive, which means a large amount of memory is needed. To implement data-intensive applications on FPGAs (Field Programmable Gate Arrays) off-chip memory is needed, which is slower (bandwidth and latency) than on-chip memory and is a potential bottleneck. A memory hierarchy should be constructed to decrease the number of off-chip transactions by reusing data stored in on-chip buffers. Therefore, the different accesses to a data element should be close together in time, i.e. exhibit a good temporal locality. Loop transformations are a means to improve the data locality by changing the execution order of computations and data accesses. This technique is commonly used for software optimizations, in particular optimization of the cache behavior. Current high-level synthesis environments for hardware design lack support to implement data-intensive applications on heterogeneous memory systems. They focus rather on parallelism than on locality.
Loop transformations not only influence the data transfers but also the control complexity of an implementation. The impact on the hardware performance can typically only be quantified after refinement to a synthesizable level. This hinders an exploration of the loop transformation space. Therefore, it would be beneficial to integrate loop transformations in high-level synthesis tools.

2. Tool Flow

First, the input C code is translated[1] into an abstract syntax tree (AST) and split into statement definitions and a polyhedral representation of the iteration domains and control structure (Fig. 2). In this polyhedral model a sequence of loop transformations can easily be applied. Only after the last transformation, the polyhedral representation is transformed back into code. With the CLooG (Chunky Loop Generator) code generator [1], C code can be generated. We have written CLooGVHDL, which adds a VHDL generation back-end to CLooG. It generates a loop controller circuit composed of communicating automata that drive the hardware implementation of the statements (Fig. 1). Different trade-offs between area and clock speed can be investigated (Fig. 3) [4].

As a test case many variants of an inverse discrete wavelet transform have been generated (e.g., Fig. 4). The results outperform those of the commercial high-level synthesis tool Impulse C and are competitive to those of the Celoxica Handel-C compiler [2,4]. In a first version of our tool, application-specific scripts were needed to translate the software description of the data path (statements) into hardware. We now have JCCI, which reads in C code, creates an intermediate representation of data and control flow and generates synthesizable VHDL for the data path. This tool will serve as a framework to develop optimization and exploration techniques.

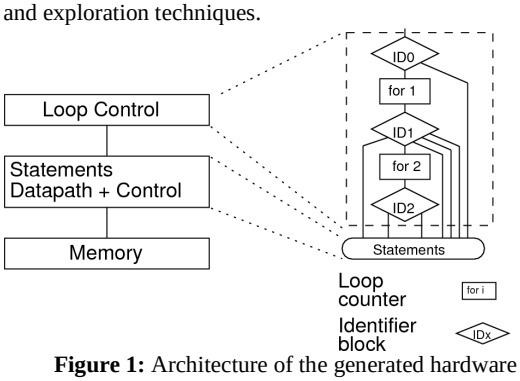

Figure 1: Architecture of the generated hardware

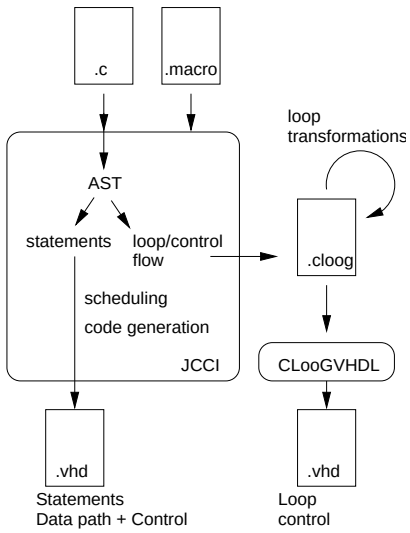

Figure 2: CLooGVHDL and JCCI tool flow

[1]The input file (.c, .macro) parsers were generated with ANTLR v3[5].

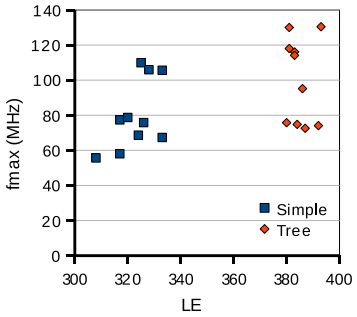

Figure 3: Speed/area exploration of a small loop controller example (LE = logic elements)

3. Extendibility with User-defined Macros

A restriction of many high-level synthesis tools is that they can deal well with the macros and corresponding hardware constructs they offer in their built-in library, but the inclusion of user made blocks has to be specified as communication with an external block. Such a block then has to fit a certain interface, possibly by adding a block that translates one interface into the other, or the communication should be described at a lower level. The former may create a hardware overhead while the latter uses the high-level input language at a level where there is little benefit over traditional hardware description languages.

With JCCI, the user can easily add its own macros which are dealt with on equal terms with the built-in macros, e.g., in the scheduling phase. The macro description input files may offer multiple implementations for a single C macro. By this, it is possible to easily swap between different implementation options without changing the C source code. For example replacing a single-cycle memory access interface by a hand-shake protocol. Thanks to the generic description of the protocols in the macro input files the scheduler will be able to pipeline the first kind of memory accesses.

4. Building a Memory System

To exploit the data locality created by loop transformations a memory hierarchy is needed. If data stored in on-chip memories is reused the number of off-chip accesses is reduced. Buffer memories may also hide the off-chip memory latency. In processor based systems a cache or scratch-pad memory performs this task. On FPGAs the freedom to build memory systems is much larger and the memory system should be targeted towards the application (Fig. 4).

In [3] we have shown how an application-specific memory system can easily be constructed step-by-step with CLooGVHDL and scripts, which were adapted for the application to translate the used macro constructs. The extendibility of JCCI with user-defined macros now offers a more systematic, generic and reusable approach for such a design flow.

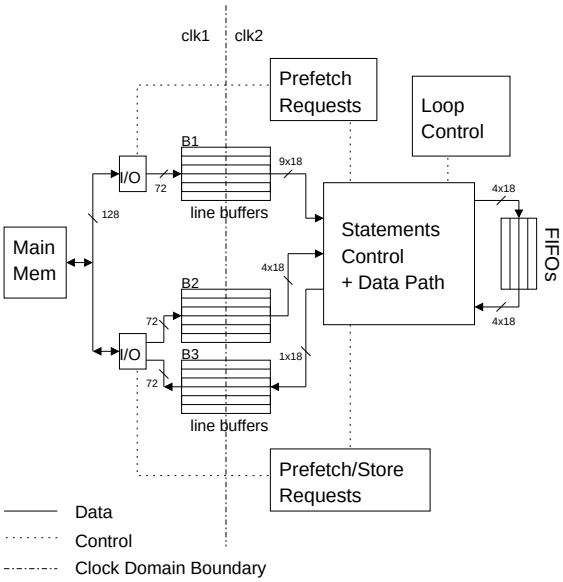

Figure 4: Line-based Inverse Discrete Wavelet Transform (IDWT) with memories after system integration [3]

5. Conclusion

This paper presented the CLoogVHDL and JCCI tool flow, which will serve as a framework to develop optimization and exploration techniques for high-level synthesis with a focus on memory systems. The extendibility with user-defined macros makes it easy to adapt a system to different IO interfaces.

6. References

[1] C. Bastoul, “Code Generation in the Polyhedral Model Is Easier Than You Think”, *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, 2004, pp. 7-16

[2] H. Devos, K. Beyls, M. Christiaens, J. Van Campenhout, E.H. D'Hollander, D. Stroobandt, “Finding and Applying Loop Transformations for Generating Optimized FPGA Implementations”, *Transactions on High Performance Embedded Architectures and Compilers*, Vol. 1(2), LNCS 4050, 2007, pp. 159-178.

[3] H. Devos, J. Van Campenhout, I. Verbauwhede, D. Stroobandt, “Constructing Application-specific Memory Hierarchies on FPGAs”, *Transactions on High-Performance Embedded Architectures and Compilers*, Vol. 3.

[4] H. Devos, “Loop Transformations for the Optimized Generation of Reconfigurable Hardware”, PhD thesis, Ghent University, February 2008.

[5] T. Parr, “ANTLR”, http://www.antlr.org/

Acknowledgement

This research is supported by the I.W.T. Grant 060068. Ghent University is a member of the HiPEAC Network of Excellence.