Hardware Generation from the Polyhedral Model

Harald Devos*,[2], Kristof Beyls*,
Mark Christiaens*,
Jan Van Campenhout*,
Dirk Stroobandt*,[1]

* *ELIS, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium*

ABSTRACT

Multimedia applications are examples of a class of algorithms that are both calculation and data intensive and have real-time requirements. As a result dedicated hardware acceleration is often needed.

Usually the on-chip memory is not sufficient to store all data and has to be extended with an external memory. The bandwidth to this memory often becomes a bottle neck. Loop transformations are needed to reduce the bandwidth, by improving the temporal and spatial data locality, and may also help to unveil the parallelism present in the algorithm. The polyhedral model offers a flexible program representation that allows to automate this kind of transformations. The class of applications that can be transformed with the polyhedral model fits very well into the class of applications that can benefit from hardware acceleration.

This paper describes how the existing tools, that generate software from a polyhedral program representation, have been extended to generate a VHDL description of a hardware controller. The corresponding data path is generated semi-automatically. Combining the generation of controller and data path creates a fast path to hardware. Our techniques enable to easily explore the design space, by generating a lot of implementation variants. After selection of a promising candidate, hand-optimizations may lead to a more optimal final implementation.

The techniques are demonstrated on an inverse discrete wavelet transform resulting in several synthesizable designs, of which one has been hand-optimized towards a FPGA implementation.

KEYWORDS: Polyhedral Model, Loop Transformation, Design Space Exploration, Hardware Synthesis, Discrete Wavelet Tranform.

1 Introduction

There exists a class of applications for which a software implementation can not offer the required performance. E.g., for real-time video decoding dedicated hardware is needed.

[1]E-mail: {hmdevos,kbeyls,mchristi,jan.vancampenhout,dstrooba}@elis.UGent.be
[2]Harald Devos is supported by the Research Foundation - Flanders (F.W.O.)

Depth 0 1 2 | | |		Statement Abstraction	Ordering vector	Iteration vector	Scheduling vector
0: **for** $i = 0, N$					
0: **for** $j = 0, N - i$					
if i = 0 **then**					
0: $B[i + j] = 1$		S1(i, j)	$(0, 0, 0)$	(i, j)	$(0, i, 0, j, 0)$
end if					
1: $B[i + j] = B[i + j]A[i][j]$		S2(i, j)	$(0, 0, 1)$	(i, j)	$(0, i, 0, j, 1)$
1: **for** $k = 0, N - 1$					
0: $C[k] = B[k] + B[k + 1]$		S3(k)	$(1, 0)$	(k)	$(1, k, 0)$

Figure 1: Program code with ordering, iteration and scheduling vector of the statements.

FPGAs (Field Programmable Gate Arrays), offer a high computational power combined with a huge internal bandwidth. Although the clock frequency is much lower than on a processor many applications can be accelerated thanks to the high parallelism available.

The design path from a high-level algorithm description to a low-level synthesizable hardware description, is a long, manual and error-prone process. Automation is needed to be able to iterate this process, e.g., to examine the influence of design choices.

Exploiting the calculative power of an FPGA is only possible when the data access rate can cope with the data process rate. Modern FPGAs contain a lot of memories and registers that are accessable in parallel and offer a huge on-chip bandwidth. However, for many applications these memories are not large enough to store all the data. The required storage can be offered by an external memory, but the bandwidth to this memory will be lower and the latency higher.

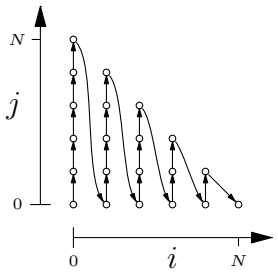
This is very similar to the *memory bottle neck*, known in processor architectures with a hierarchy of main memory, caches and registers.

To reach a high performance the accesses to the slower external memory have to be minimized. Therefore the data locality has to be optimized. This is done by loop transformations that change the execution order of statement instances in a loop nest. Although these techniques are mainly developed targeting software implementations they are also useful for designing hardware.

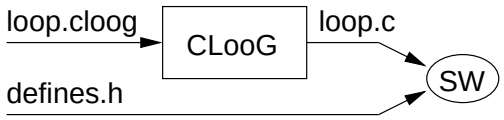
A polyhedral program representation can be used to automate loop transformations. A VHDL back-end was written to partially automate the hardware design process.

2 The Polyhedral Model

A part of a program where the control flow is independent of the processed data is called a SCoP (Static Control Part), and can be represented in the polyhedral model. The iteration domain of each statement is described as a union of convex sets of points in $\mathbb{N}^M$, i.e., a polyhedron (Fig.2(a)), where M is the number of iterators. By extending the iteration vector to a vector with dimension $2M + 1$, called the scheduling vector, the ordering of statements can be described and manipulated (Fig. 1). Loop transformations are reduced to linear transformations on the polyhedrons and on the ordering vectors.



(a) Polyhedral representation of the domain of $S2$ (Fig. 1). The arrows indicate the execution order.



(b) CLooG (Chunky Loop Generator), translates a polyhedral representation (*.cloog) back into C-code.

Figure 2: Polyhedral representation and path to software.

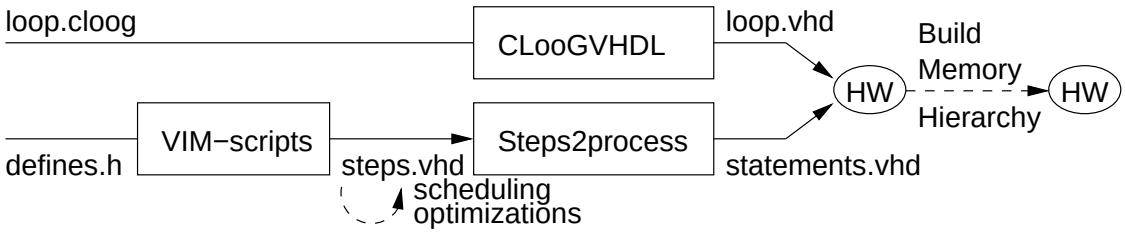


Figure 3: Generating hardware from a polyhedral representation. The dashed lines indicate manual intervention.

This representation overcomes a lot of limitations of syntactic representations and facilitates the composition of a sequence of transformations [Cohe05]. Only after all loop transformations are performed, the polyhedral representation is translated back into an AST to obtain executable code [Bast03] (Fig. 2(b)). The definitions of statements remain unchanged. Only the control structure defining the execution order changes.

A lot of techniques for optimizing software for (multi-)processors use this model but it is rarely used with the purpose of generating hardware.

3 Hardware Generation

The techniques described above are not only useful for optimizing software but also for hardware. Therefore the CLooG tool was extended with a VHDL back-end, CLooGVHDL. A control hardware structure composed of communicating automata is generated that should be connected with hardware that implements the statement functions (Fig. 3). More details can be found in [Devo06].

4 Case Study: the 2-Dimensional Inverse Discrete Wavelet Transformation

The toolchain was tested by implementing an Inverse Discrete Wavelet Transform (IDWT). The loop transformations were performed by the WRaP-IT/URUK tool set, that allows to

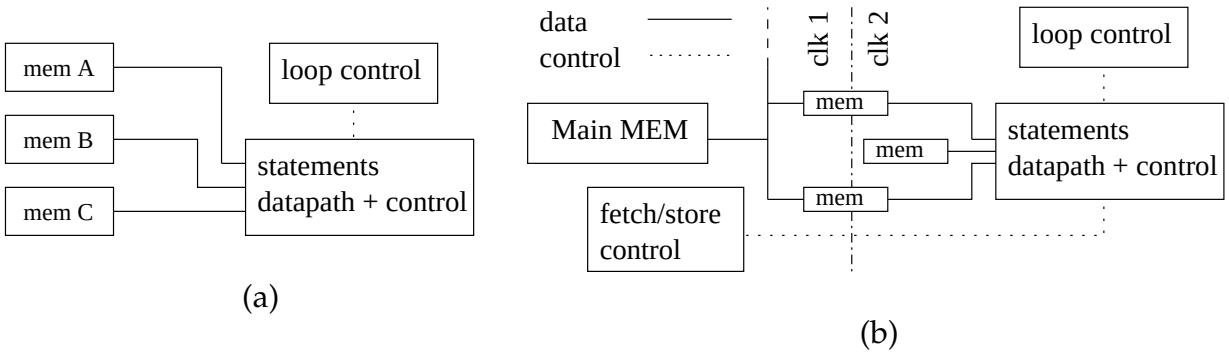


Figure 4: Hardware architecture before and after adding a memory hierarchy.

describe sequences of transformations in a script [Girb05, Cohe05]. From the several generated variants, one was chosen to be extended manually with a memory hierarchy (Fig. 4). Prefetching and storing is done in bursts and in parallel with the execution of the statements. This way the obtained data locality could be exploited what led to a large performance increase in comparison with an untransformed manual design [Devo04].

References

[Bast03] C. BASTOUL. Efficient code generation for automatic parallelization and optimization. In *ISPDC'2 IEEE International Symposium on Parallel and Distributed Computing*, pages 23–30, Ljubljana, october 2003.

[Cohe05] A. COHEN, S. GIRBAL, D. PARELLO, M. SIGLER, O. TEMAM, AND N. VASILACHE. Facilitating the search for compositions of program transformations. In *ACM Int. Conf. on Supercomputing (ICS'05), Boston, Massachusetts.*, June 2005.

[Devo04] H. DEVOS, H. EECKHAUT, B. SCHRAUWEN, M. CHRISTIAENS, AND D. STROOBANDT. Ever considered SystemC? In *Proceedings of the 15th ProRISC Workshop*, pages 358–363, Veldhoven, November 2004.

[Devo06] H. DEVOS, K. BEYLS, M. CHRISTIAENS, J. VAN CAMPENHOUT, E. D'HOLLANDER, AND D. STROOBANDT. Finding and Applying Loop Transformations for Generating Optimized FPGA Implementations. *Transactions on High Performance Embedded Architectures and Compilers*, 2006. To be published.

[Girb05] S. GIRBAL. *Optimisations d'applications - Composition de transformations de programme: modèle et utils*. PhD. Thesis, Université de Paris-Sud, 2005.