Loop transformations for generating scalable hardware

Harald Devos

Promoter(s): Dirk Stroobandt, Jan Van Campenhout

Abstract—**Multimedia applications emerge on all kinds of devices, from small mobile systems up to desktop computers with varying ranges of quality, resolution, power and other requirements. These and other data intensive applications often need hardware acceleration in order to satisfy real time constraints. Designing application specific hardware for all different platforms and all different combinations of quality parameters is a huge task. Therefore techniques should be developed to automatically generate a lot of different versions of hardware adapted to the parameters of the application and platform.**

The memory bandwidth is often a bottleneck for these kind of applications. Loop transformations are used to improve the memory access pattern. Most research focuses on optimizing SW towards (multi-)processor architectures. The question is how the developped techniques can be used or adapted with the purpose of generating variants of Pareto-optimal hardware where the memory hierarchy can be freely chosen.

Keywords—**polyhedral model, transformations, FPGA**

I. Introduction

MULTIMEDIA systems are everywhere, from small mobile phones up to desktop computers and home cinema systems. They run similar applications on a huge range of platforms, all with a different screen size, calculation power, memory size and bandwidth.

They are part of a large class of signal processing systems that often need hardware acceleration. The QoS (Quality of Service), is in relation with the available hardware.

Designing the hardware for all different platforms is a huge task. The occurance of reconfigurable hardware (e.g. FPGAs), that can be configured with different hardware designs (e.g. each with different QoS, power and area consumption) makes the number of designs to be made even higher. Tools have to be developped to generate different scaled versions of a design.

Most of the algorithms are not only computationally intensive but also data intensive. The memory bandwidth and latency can become a bottleneck. Transformations are needed to improve the spatial and temporal locality of the memory accesses. We will focus on programs where most of the execution time is spent in a small part of the code, consisting of a set of nested loops. Almost all signal processing algorithms contain such *hot* loop nests.

II. 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) Such SCoPs where the loop bounds are linear functions of the iterators of the enclosing loops and some global parameters can be represented in the polyhedral model. For each statement the iteration domain is described as a set of lineair inequalities in the iterators and the parameters. They can be seen as polyhedrons (convex sets of points in a lattice) in the N-dimensional space (where N is the number of iterators). Loop transformations are reduced to linear transformations on the polyhedrons of the statements and on the scheduling matrix which defines the ordering of the statement instantiations.

This representation overcomes a lot of limitations of syntactic representations and facilitates the composition of a sequence of transformations[1]. A lot of techniques have been developped for optimizing software for (multi-)processors but the model is rarely used with the purpose of generating hardware.

We would like to investigate whether the existing optimization algorithms for software can be used, in an adapted form, with the purpose of generating hardware and which other transformations are needed. We also believe that transformations in the polyhedral model can be used with the goal of making scaled versions of a design.

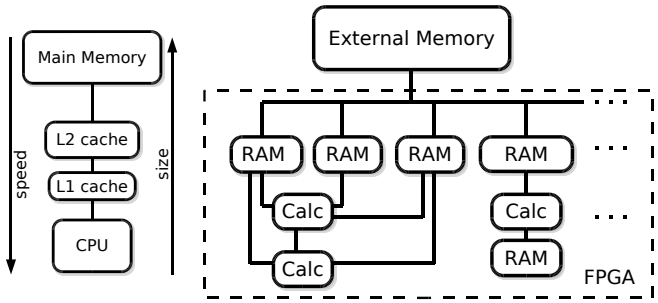

Fig. 1. Comparison between processor and a possible FPGA architecture.

III. Difference between processor and FPGA/ASIC architecture

The optimization criteria that are used for finding good loop transformations depend on the used architecture. This section describes the main differences between a general purpose processor architecture and an application specific architecture, e.g. implemented on a FPGA[1] board. (Fig.1)

A. Memory

The memory hierarchy of a processor is more or less linear. There is one large main memory and smaller but faster caches closer to the processor and finally some registers in the processor.

H. Devos is with the Electronics and Information Systems Department, Ghent University (UGent), Gent, Belgium. He is supported by the fund for Scientific Research Flanders(F.W.O.). E-mail: Harald.Devos@UGent.be .

[1]We will from here on only consider FPGAs because all FPGA architectures can also be implemented with an ASIC.

All modern FPGAs contain a lot of memory blocks that can be connected to form smaller or larger, single or multiple port memories. All these memories can be accessed in parallel within one clock cycle. A larger but slower memory is placed next to the FPGA. To increase the bandwidth to the external memory, memory transactions should be done in bursts. RAM blocks can be used as buffers between the burst transactions (cf. prefetching[2]) and the computation driven consumption and production of data.

It is clear that both a software and a hardware implementation will benefit from spatial and temporal locality of the data accesses however there are a lot of differences. Software optimization transforms the data access pattern. On an FPGA it becomes also possible to adapt the memory hierarchy to the data access pattern.

B. Parallelism

Processors are sequential in origin. Parallelism is introduced on a very small scale, e.g. instruction pipelining (invisible to the programmer), or at a large scale, e.g. threads running on parallel processors.

The power of FPGAs lies in their massive amount of parallelism. All available hardware blocks (memories, multipliers, look up tables,...) can potentially work in parallel. The designer can make a tradeoff between used hardware resources and execution time by choosing what to do sequential and what in parallel.

C. Control flow

On a processor the software code controls the execution order of operations. Control instructions and operations on data are executed (sequentially) on the same processor.

On an FPGA the control is implemented by interconnections and finite state machines. The data and control flow can be separated and work in parallel.

Transformations to decrease the amount of misses in the instruction cache are useless when generating hardware. Control dominated applications will run faster on a processor as there is little parallelism to be exploited.

IV. Progress

First an implementation of an Inverse Discrete Wavelet Transform on a FPGA board was made by hand [3][4]. This way the strengths and weeknesses of the platform and the needed code transformations for optimization could be discovered. The memory bandwidth was indeed the bottleneck. Also the opportunities for scalability were investigated.

The current step is to create a traject from software (SCoP) to hardware (described in e.g. VHDL). Tools to translate SCoPs to a polyhedral representation and do transformations with it allready exist[1]. Cloog[5], a library generating software starting from a polyhedral representation of a SCoP (without statement information), was extended to generate VHDL. This way hardware can be made for the control part of the design in the case of a sequential execution of the statement instantiations. The datapath and memory hierarchy still have to be generated manually.

The next step consists of creating a tool to generate VHDL for the hardware implementing the statements and memories. Also the possibility of creating parallelism should be added to the tools. Once a toolchain from software to hardware exists the effects of loop transformations on hardware performance can be investigated more easily. The development of hardware optimizing algorithms could be a result. The tool chain will also make it possible to create scaled versions of a design. Existing techniques to create hardware starting from a software description, have difficulties with memory accesses, e.g. SPARK[6], or support only a restricted set of IO structures (e.g. ImpulseC).

V. Conclusion

A lot of research has been done on techniques to improve the performance of software on a processor. Some techniques are also usefull for increasing the performance of dedicated hardware, but they are not sufficient to deal with the huge amount of possibilities hardware design offers.

The polyhedral model has been used to facilitate doing loop transformations on programs. The first steps to extend its application area towards creation of (scalable) hardware were taken.

Acknowledgments

Harald Devos is supported by the fund for scientific research Flanders (F.W.O.).

References

[1] Albert Cohen, Sylvain Girbal, David Parello, Marc Sigler, Olivier Temam, and Nicolas Vasilache, "Facilitating the search for compositions of program transformations.," in *ACM Int. Conf. on Supercomputing (ICS'05), Boston, Massachusetts.*, June 2005.

[2] Steven P. Vanderwiel and David J. Lilja, "Data prefetch mechanisms," *ACM Comput. Surv.*, vol. 32, no. 2, pp. 174–199, June 2000.

[3] D. Stroobandt, H. Eeckhaut, H. Devos, M. Christiaens, F. Verdicchio, and P. Schelkens, "Reconfigurable hardware for a scalable wavelet video decoder and its performance requirements," *Computer Systems: Architectures, Modeling, and Simulation*, vol. 3133, pp. 203–212, July 2004.

[4] Harald Devos, Hendrik Eeckhaut, Benjamin Schrauwen, Mark Christiaens, and Dirk Stroobandt, "Ever considered SystemC ?," in *Proceedings of the 15th ProRISC Workshop*, Veldhoven, November 2004, pp. 358–363.

[5] C. Bastoul, "Code generation in the polyhedral model is easier than you think," in *PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques*, Juan-les-Pins, september 2004, pp. 7–16.

[6] Sumit Gupta, Nick Savoiu, Nikil Dutt, Rajesh Gupta, and Alex Nicolau, "Using global code motions to improve the quality of results for high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,*, vol. 23, no. 2, pp. 302–312, February 2004.

[7] "The RESUME project: Reconfigurable Embedded Systems for Use in Scalable Multimedia Environments," http://www.elis.UGent.be/resume.