Fast SW/HW Co-design framework for real-time image/video processing

Chao Li, Souleymane Balla-Arabe, Vincent Brost, Fan Yang
Université de Bourgogne Franche-Comté
Faculté Mirande, 21000 Dijon

Abstract

Real-time digital image/video processing is a widely used computer vision/graphics technique. For the purpose of high performance designs, variant computation platforms are make available to the engineers with affordable prices. However, these sophisticated devices usually require a specific implementation framework to transplant the desired algorithm from software environment into hardware environment. Since this process is quite effort-consuming, finding a new approach that can accelerate the development cycles the image/video processing applications becomes a new challenge. This work focus on fast SW/HW Co-design Framework for real-time image/video processing. After a carefully methodology exploration to the currently available hardware platforms, we select Field Programmable Gate Arrays (FPGAs) as our target platform due to its high performances in terms of efficiency-cost. Next, we base our research on an improved C-synthesis design flow, and development a novel Code and Directive Manipulation Strategy for High-Level Synthesis (CDMS4HLS). Finally, intensive experiment demonstrate that, compared with the other similar approaches, our method can essentially reduce the effort of the development cycles and potentially improve the performances of final implementations.

1. Introduction

In image science, image/video processing is one of the applications of signal processing technics. It includes methods for analyzing, processing or treating images. The motivation of this field is to improve the target images to meet the requirements in visual, mental or other technical terms. Since images are usually stored in digital format currently, image/video processing refers to digital image/video processing in most cases.

With the fast development of digital signal processing technics, image/video processing is widely used in variant fields, such as satellite image analyzing, computer-aided medical diagnosis, face identification, microscope image processing and car barrier detection, etc.. Meanwhile, the algorithms for image processing become increasingly complex and computationally intensive. For the purpose of performances, series of computation platforms are developed and made available with an affordable price, i.e. Graphics Processing Unit (GPU), Digital Signal Processor (DSP), Centeral Processing Unit (CPU) and FPGA etc.. Since these sophisticated devices have quite different architectures, engineers have to transplant their designs from software environments (i.e. Matlab, OpenCV and C/C++) into specific hardware environments within different development frameworks, i.e. Message Passing Interface (MPI) for CPU, Compute Unified Device Architecture (CUDA) for GPU or Hardware Description Language (HDL) for FPGA etc..

Implementing a design from software environment into hardware environment is a painful and effort-consuming process. This is because the languages used for hardware configuration are usually low-abstract, which are inconvenient for algorithm specification. In order to improve both productivity and performances of the desired designs, many image research and development communities base their works on a streamlined design flow, known as SW/HW Co-design.

SW/HW Co-design is an important approach to ensure an efficient final implementation of the product. In order to further improve the research and development productivity, series of Computer Aided Software Engineering (CASE) based software tools are put into operation. Moreover, the progresses of software engineering increasingly provide opportunities to seek new design methodology with better productivity.

We focus our work on the SW/HW Co-design methodology exploration for real-time image/video processing development. The main purpose of this research project is to propose a new methodology to quickly conceive and realize high-performance implementations for real time image/video processing with high adaptability capacity. According to the achievements of a careful bibliography study, we base the new approach on the High-Level Synthesis (HLS) SW/HW Co-design framework for FPGAs, and improve it according a source-to-source compiler. During this effort, a novel CDMS4HLS is developed.

In this new SW/HW Co-design framework, we provide multiple advantages:

- The new approach provides a C/C++ development environment, which allows engineers to realize rapid prototyping of signal and image processing on target devices ignoring completely hardware aspects.
- The new approach performs an open and adaptive framework which allows parallel optimizations in different levels, i.e. FLP, LLP, ILP and DLP etc.. Nevertheless, the optimizations can easily made and don't require much hardware knowledge.
- Comparing with the conventional methodologies, the new approach doesn't make any negative effects to the designs in terms of power-efficiency, and the loss of accuracy or cost-efficiency due to the hardware devices are acceptable in most cases if it exists.

Experiment results demonstrate that our approach can provide the engineers a quite software-convenient environment for development and optimization as well. Furthermore, our approach can make more efficient use of the hardware resources than the other similar FPGA design flows and provide higher performances than the implementations on the other computation platforms.

After the introduction section, the remains of this thesis is organized as follows: Section 2 explores the state of the art of HLS based FPGA design flows, Section 3 presents the proposed code and directive manipulation strategy for it, Section 4 discusses the experiment results and a conclusion is given in the final section.

2. Related works

Over the past twenty years, High-Level Synthesis (HLS) technique has made great progress. Recently, some robust and mature HLS SW/HW Co-design development frameworks have been made available to engineers, i.e. Vivado_HLS of Xilinx [1] and Catapult C Synthesis Work Flow [2]. These convenient tools allow to specify targeted hardware behaviour in high abstract levels rather than RTL, and then create the Hardware Description Language (HDL) specification of desired FPGA implementations from its software prototype through an automatical C-to-RTL transformation. This approach can greatly accelerate the developments by freeing the designers from the boring work of hardware implementing [3].

Fig.1-(a) illustrates the framework of HLS-based FPGA designs. First of all, designers specify the software prototype of targeted algorithm in C-like languages and debug it in a test bench using common C compilers. Next, the confirmed code is imported into a HLS tools as original sources for C-to-RTL synthesis. During this process, the designers configure the synthesis constraints/directives to make

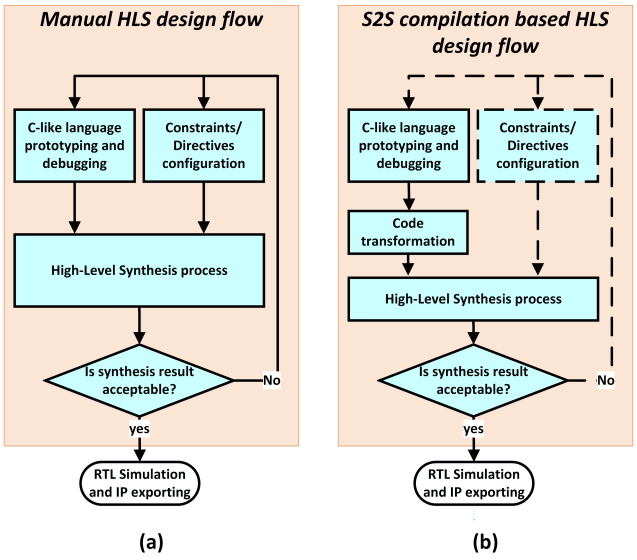

Figure 1: Manual and source-to-source compiler based HLS design framework.

their implementations suitable for different design requirements. At last, the generated HDL specification is simulated by a RTL simulator like ModelSim, and then exported as IP-blocks. This approach doesn't require specific knowledge of both software and hardware, so users can concentrate their attentions only on the algorithm specifications in high abstract levels. However, with the widespread of HLS tools in the ES (Embedded System) world, more issues related to time control, execution speed and consummation etc. emerge. In order to find out the best design solution, designers have to configure repeatedly the directives, and sometimes even have to re-specify their algorithms to ensure the input sources to be detectable by HLS processes. This is a quite painful and effort-costly job even for an experienced SW/HW Co-design engineer.

The issue of HLS design flow discussed above is caused by three major reasons: a) the C-language suitable to HLS tools is just a subset of C-like languages, we therefore cannot benefit to all the C advantages during the algorithm specification, b) different designers and algorithms may result in different code structures and styles, so the data dependency of input sources sometimes can't be perfectly determined by scheduling process for optimization and c) the existing synthesis tools provide dozens of directives for hardware constraints which requires the designers to be quite familiar with synthesis tools. For the purpose to further simplify the development cycles, a source-to-source (S2S) transformation based HLS design flow was developed (see Fig.1-(b)). In this new framework, a S2S compiler is inserted between the software prototyping and synthesis process. In contrast with the manual specification and

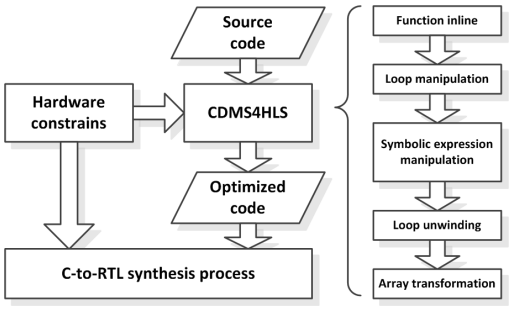

Figure 2: CDMS4HLS compilation process.

configuration, this bridge tool automatically transforms the original code into the sources more efficient. For example, Alle et al. propose an efficiency improvement approach for loop pipelining in HLS through a semi-automatic source-to-source transformation in [4].

3. souce-to-source compilation based HLS design flow

C-to-RTL synthesis is the key technique of HLS. It offers many opportunities to optimize the designs in different hierarchies, including function level, loop level, instruction level and data level etc.. Therefore how to perform an efficient source code and perfectly combine these optimization strategies together becomes a new challenge for engineers. This is usually a painful and time-costly work because it has to be repeated several times until an acceptable, even if the best solution is found. Thus, some C-to-C compilers emerges. That effectively raise the productivity of such design by automatically improving the efficiency of manual code.

This section describes a C-to-C compilation strategy for HLS, CDMS4HLS, which can streamline the code optimization process. Unlike some existing research productions, we base this work on the special characteristics of HLS in terms of synthesis process rather than borrow some achievements from other efforts about parallel computing for high performance computers. The over-all structure of CDMS4HLS is shown in Fig.2, which consists of function inline, loop fusion, symbolic expression manipulation, loop unwinding and array reshape. These 5 steps are effected on input source code in a proper order for the purpose to enable each optimization method to make a maximum effectiveness.

4. Experiments

We compare the proposed approach with other two functionally similar design flows (see Fig.3) using the source code of four basic image processing algorithms, including 3×3 filter for RGB images, matrix product (MatPro), Image Segmentation using Sobel operator (ImSeg) and Stereo Matching using sum of squared difference (StMatch). We base the first design flow on two improved conventional source-to-source C/C++ compilers: an improved PoCC polyhedral framework [5]–[7] and the Generic Compiler Suite (GeCoS) [4], [8], [9] (defined as PolyComp), while the other one on the Vivado_HLS Design Suite [10] (defined as Vivado_HLS). In order to obtain an unbiased conclusion, all the source codes are synthesized using AutoESL and their data formats are normalized to 32-bit integer numbers. Considering that PolyComp does not have the ability of I/O interface manipulation, we set the I/O protocol of the target implementations as the default of the HLS tool used.

The latency speedups of the three design flows with different algorithms are compared in Fig.4. This result is normalized to the int_32 original versions of the related algorithms. These three approaches respectively achieve an

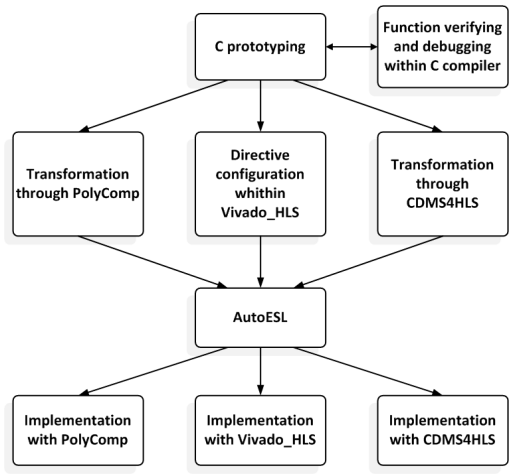

Figure 3: Implementing flow with different code optimization methods, including PolyComp, manual directive configuration within Vivado_HLS and CDMS4HLS.

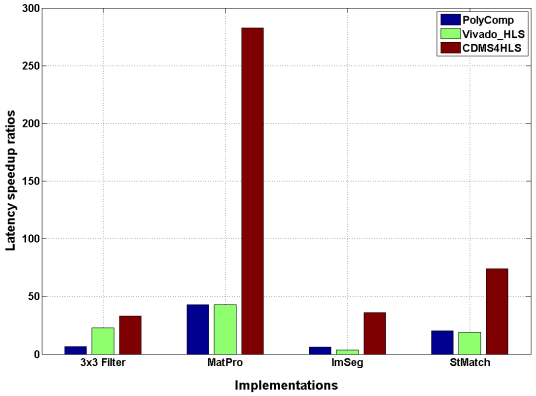

Figure 4: Latency speedup comparison.

average of 19.01×, 22.19× and 106.54× speedups. This demonstrates that the proposed approach can gain more performance improvements in terms of latency consumption. Compared with the other designs, CDMS4HLS has the ability to manipulate the source code in a lower instructions level, which provide more optimization opportunities to HLS tools. Furthermore, our method can effectively reduce the transition number of the FSM behaviours of the target implementations. For example, the transition number of the MatPro optimized by CDMS4HLS is only half as many as PolyComp and Vivado_HLS respectively. In additional, it should note that the acceleration gains due to the interface expending are not taken into account. That is, CDMS4HLS and Vivado_HLS may achieve more speedups than PolyComp.

5. Conclusion

This paper presents a novel source-to-source compilation strategy for High-Level Synthesis. Unlike the other studies, the features of HLS procedure and its existing tools are studied in detail. Basing on these efforts, we designed a customized code and directive manipulation strategy (CDMS4HLS) for it.

The proposed approach improves the performances of the desired designs in various ways, including function and loop hierarchy optimization, symbolic expression manipulation and memory/interface protocol manipulation. In the experiments, we evaluate our approach using four basic algorithms and compare it with two other similar design flows: PolyComp and Vivado HLS. The results demonstrate that CDMS4HLS is an effective code optimization strategy which improves substantially the HLS based FPGA designs.

For the future work, we plan to apply the FPGA design flow improved by the proposed method into some complex real-time image processing applications. Since computationally intensive algorithm may result in a complicate control flow and operation scheduling, more efforts are usually required for development and optimization. Testing and verifying CDMS4HLS according to a practical and complex case can further evaluate its feasibility for commercialization. Meanwhile, we hope that the efforts of this paper can bring some enlightenments to the studies for fast FPGA development framework.

References

[1] *Vivado design suite user guide*, UG902(2012.2), XILINX, Jul. 2012 (cit. on p. 2).

[2] G. Wang, *Catapult c synthesis work flow tutorial*, Version 1.3, ECE Department, Rice University, Oct. 2010 (cit. on p. 2).

[3] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for fpgas: from prototyping to deployment," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 4, pp. 473–491, 2011, ISSN: 0278-0070. DOI: 10.1109/TCAD.2011.2110592 (cit. on p. 2).

[4] M. Alle, A. Morvan, and S. Derrien, "Runtime dependency analysis for loop pipelining in high-level synthesis," in *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, May 2013, pp. 1–10 (cit. on p. 3).

[5] W. Zuo, Y. Liang, P. Li, K. Rupnow, D. Chen, and J. Cong, "Improving high level synthesis optimization opportunity through polyhedral transformations," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '13, Monterey, California, USA: ACM, 2013, pp. 9–18, ISBN: 978-1-4503-1887-7. DOI: 10.1145/2435264.2435271. [Online]. Available: http://doi.acm.org/10.1145/2435264.2435271 (cit. on p. 3).

[6] "Pocc. the polyhedral compiler collection.," in. [Online]. Available: http://www.cs.ucla.edu/~pouchet/software/pocc/ (cit. on p. 3).

[7] P. Li, L.-N. Pouchet, and J. Cong, "Throughput optimization for high-level synthesis using resource constraints," in *IMPACT 2014. Fourth International Workshop on Polyhedral Compilation Techniques. In conjunction with HiPEAC 2014*, Vienna, Austria, Jan 20, 2014 (cit. on p. 3).

[8] A. M. Steven Derrien and A. Kumar, "S2s4hls-sp1 progress report," INRIA - University of Rennes 1, INRIA - ENS Cachan and INRIA - LIP, Tech. Rep., 2008 (cit. on p. 3).

[9] A. Morvan, S. Derrien, and P. Quinton, "Efficient nested loop pipelining in high level synthesis using polyhedral bubble insertion," in *Field-Programmable Technology (FPT), 2011 International Conference on*, Dec. 2011, pp. 1–10. DOI: 10.1109/FPT.2011.6132715 (cit. on p. 3).

[10] *Vivado design suite tutorial*, UG871(v2012.2), XILINX, Feb. 2012 (cit. on p. 3).