Journal of Systems Architecture 61 (2015) 646–658

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Synthesis and optimization of image processing accelerators using domain knowledge

Oliver Reiche*, Konrad Häublein, Marc Reichenbach, Moritz Schmid, Frank Hannig, Jürgen Teich, Dietmar Fey

Department of Computer Science, Friedrich-Alexander University Erlangen-Nürnberg (FAU), Germany

ARTICLE INFO

Article history:
Received 30 April 2015
Revised 31 August 2015
Accepted 25 September 2015
Available online 9 October 2015

Keywords:
Hardware accelerators
Image processing
Synthesis
Code generation

ABSTRACT

In the domain of image processing, often real-time constraints are required. In particular, in safety-critical applications, timing is of utmost importance. A common approach to maintain real-time capabilities is to offload computations to dedicated hardware accelerators, such as Field Programmable Gate Arrays (FPGAs). Designing such architectures is per se already a challenging task, but finding the right design point between achieving as much throughput as necessary while spending as few resources as possible is an even bigger challenge.

To address this design challenge in the domain of image processing, several approaches have been presented that introduce an additional layer of abstraction between the developer and the actual target hardware. One approach is to use a Domain-Specific Language (DSL) to generate highly optimized code for synthesis by general purpose High-Level Synthesis (HLS) frameworks. Another approach is to instantiate a generic VHDL IP-Core library for local imaging operators. Elevating the description of image algorithms to such a higher abstraction level can significantly reduce the complexity for designing hardware accelerators targeting FPGAs. We provide a comparison of results for both approaches, a non-expert algorithm developer can achieve. Furthermore, we present an automatic optimization process to give the algorithm developer even more control over trading execution time for resource usage, that could be applied on top of both approaches. To evaluate our optimization procedure, we compare the resulting FPGA accelerators to highly optimized Graphics Processing Unit (GPU) implementations of several image filters relevant for close-to-sensor image and video processing with stringent real-time constraints, such as in the automotive domain.

1. Introduction and related work

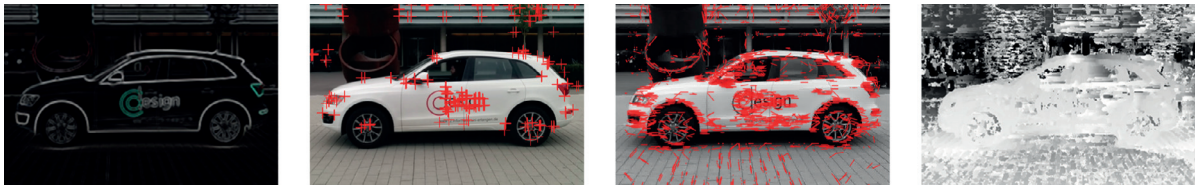
Today, image processing is a vital part of many application domains. Typical examples include smart home or home automation, where the building uses computer vision to recognize its inhabitant and provide personalized comfort. Another well-known example is medical imaging, where sensor noise reduction, visual enhancement of radiographs, or the precise automatic detection of the patient's position for radiant exposure is of high importance. Especially in the automotive domain, computer vision is largely applied to implement advanced driver assistance systems or even autonomous driving. These computer vision algorithms are mostly based on low-level image processing operations for feature extraction, such as the Sobel operator for edge detection, the Harris corner detector, the census transform for detecting the optical flow, or simple block matching to obtain a depth map for stereo vision, depicted in Fig. 1.

In particular in the automotive domain, the questions arises where to compute those image filters, as many different compute devices are already integrated into modern cars. Depending on the filter's complexity, possible targets are an already existing Electronic Control Unit (ECU), a dedicated microcontroller solely for that purpose, an embedded Central Processing Unit (CPU) or GPU, or even an FPGA. The right choice depends on many factors, such as development effort, power efficiency, area constraints, and real-time capabilities, which are most important for safety-critical systems.

In order to meet strict real-time constraints, the traditional way, that an image sensor just captures image data and transfers it to a processing system is often not sufficient. Rather, the data has to be processed where the information is acquired, which means in or near the image sensor. This leads to a new class of devices, called *smart cameras* [1].

* Corresponding author. Tel.: +49 91318567270.
E-mail addresses: oliver.reiche@cs.fau.de (O. Reiche), konrad.haeublein@cs.fau.de (K. Häublein), marc.reichenbach@cs.fau.de (M. Reichenbach), moritz.schmid@cs.fau.de (M. Schmid), hannig@cs.fau.de (F. Hannig), teich@cs.fau.de (J. Teich), dietmar.fey@cs.fau.de (D. Fey).

http://dx.doi.org/10.1016/j.sysarc.2015.09.004
1383-7621/



(a) Edge detection (b) Corner detection (c) Optical flow (d) Stereo vision

Fig. 1. Typical image filters for feature detection in the automotive domain.

One of the first smart cameras was developed by the group of Wolf [2]. They used a Trimedia CPU for image preprocessing tasks. To achieve higher frame rates, they proposed to extensively use SIMD[1] units. Other approaches, such as the one described in [3], use Digital Signal Processors (DSPs) to achieve a very high computing power. Here, a scalable system that consists of up to 10 DSPs for parallel processing is built to further increase performance. Even more customized architectures have been developed. For example, in [4], a dedicated integrated circuit was developed to speed up image processing within smart cameras. A good survey of smart camera approaches is provided in [5].

With the emerging technology of FPGAs, these devices have been quickly adopted for the design of smart camera systems. One big advantage is the number of parallel processing units, which can be instantiated in FPGAs as 1D or 2D arrays since image processing algorithms are in general well parallelizable [6]. Therefore, many new architectures were created on the basis of FPGAs in the past years.

It is well-known that application-specific hardware might give the highest performance and most efficient resource utilization. On the contrary, application-specific development is a time consuming and error prone task. One step towards mastering this challenge are HLS tools, such as Calypto's Catapult, NEC's CyberWorkBench [7], Impulse's CoDeveloper C-to-FPGA tools, the Riverside Optimizing Compiler for Configurable Computing (ROCCC 2.0) [8], or Vivado HLS from Xilinx (formerly known as AutoPilot [9]). However, all these HLS tools can be considered as general purpose frameworks and do not make use of domain-specific knowledge for image processing. The *smart buffer* concept of ROCCC is similar to our buffering approach to exploit data reuse as much as possible. However, ROCCC does not provide any domain-specific extensions, such as local operators for image processing. Sometimes, HLS frameworks include specific libraries to provide elemental architecture constructs and filtering implementations, e.g., the partial port of the OpenCV library [10] for Vivado HLS from Xilinx.

Schmid et al. proposed in [11] a pipeline design for range image preprocessing on FPGAs. Here, several filters for compensating sensor deficiencies (e.g., noise and pixel defects) were designed by using the HLS framework PARO [12] and evaluated in an experimental setup, consisting of a Microsoft Kinect and Xilinx Virtex-6 LX240T FPGA.

One of the first language approaches has been presented by Böhm et al. with the single assignment programming language SA-C [13] for targeting image processing applications on reconfigurable systems. Instead of utilizing a DSL, algorithms are provided as an imperative high-level C-based description.

Serot et al. introduced CAPH [14], an actor-based DSL for stream-processing applications, described as networks of dataflow. They target smart camera system by generating SystemC and VHDL code before running the actual synthesis. Similarly, Janneck et al. propose the actor language CAL [15] for FPGAs, as well as OpenDF [16], a toolset based on the dataflow programming model, presented by Bhattacharyya et al. All approaches have in common to focus on video encoding applications with a strong orientation towards hardware synthesis and ASIC architectures.

Another DSL for generating hardware accelerators is Darkroom [17], introduced by Hegarty et al. They propose functional programming to describe local image operators, which are translated into line-buffered pipelines. Besides ASICs and FPGAs, also CPU implementations can be generated. Unlike Halide [18], which Darkroom based on, targeting GPUs is not supported and the domain is restricted to only static, fixed size windows, or stencils.

A further approach is taken by the HIPA^{cc} framework [19] to generate code for FPGA HLS. HIPA^{cc} is a publicly available framework[2] for the automatic code generation of image processing algorithms for GPU accelerators. Starting from a C++ embedded DSL, HIPA^{cc} delivers tailored code variants for different target architectures. In this way, code generation offers true portability of programs and performance across different platforms, and delivers increased productivity, as developers need not be concerned about implementation details, but can focus on functionality [19]. Recently, HIPA^{cc} was extended to also be able to generate C++ code for the C-based HLS tool Vivado HLS [20], even capable of handling complex multiresolution applications [21].

In contrast to applications for high-performance computing, hardware accelerators for FPGAs may be subject to contrasting design goals, such as a high processing speed or low use of the available hardware resources. A throughput optimized implementation most likely consumes the highest amount of resources. Lowering the throughput, however, provides an opportunity for resource sharing, which may significantly reduce the amount of required resources. Especially for close-to-sensor processing, the resources available for the accelerator implementation may be limited, yet at the same time, a lower throughput might suffice. In this work, we therefore present an automatic design space exploration and design optimization procedure, which can incrementally refine a hardware implementation to achieve a specific design goal.

The contributions we present in this work can be summarized as follows:

- A comparison of two abstract domain-specific approaches for the fast generation of FPGA implementations for local image operators.
- An optimization feedback loop that could be applied on top of both approaches to meet certain design constraints.
- An analysis of resource savings due to the optimization process for image filters, relevant to close-to-sensor image and video processing with stringent requirements (e.g., imaging and computer vision algorithms used in advanced driver assistance systems), compared with the achievable throughput of embedded and server-grade GPUs.

The remainder of this work is organized as follows. First, we highlight the domain of image processing with local operators in Section 2, before introducing both approaches in detail in Sections 3 and 4. The applied optimization loop, the optimization heuristics, and the reasoning for their design are presented in Section 5.

[1] SIMD: *Single Instruction, Multiple Data*, according to M. Flynn's taxonomy.

[2] http://hipacc-lang.org .