IPOL - A Domain Specific Language for Image Processing Applications

Christian Hartmann, Marc Reichenbach, Dietmar Fey
Chair of Computer Architecture
Friedrich-Alexander University Erlangen-Nürnberg (FAU),
Martensstr. 3, 91058 Erlangen, Germany
{christian.hartmann,marc.reichenbach,dietmar.fey}@cs.fau.de

***Abstract*—In recent years, the use of image processing systems has increased steadily. However, most of them are very complex and contain several tasks with different complexities which result in varying requirements for computing architectures. Nevertheless, a general processing scheme in every image processing application has a similar structure, called image processing pipeline: (1) capturing an image, (2) pre-processing using local operators, (3) processing with global operators and (4) post-processing using complex operations. Therefore, application-specialized hardware solutions combined in a heterogeneous system are used for image processing. To archive this, finding an optimal heterogeneous hardware architecture to meet the image processing application requirements is the central problem and still unsolved. Instead, engineers use languages like VHDL, Verilog, C/C++ and Cuda for designing such systems. But, these kind of languages are not suitable for system analysis - they provide a hardware specific solution for a specific algorithm. Therefore, a holistic modeling of a complete image processing pipeline, with automatic optimization and assignment to different heterogeneous computing cores is not possible. To overcome this problem, we propose in this paper a new domain specific language, called Image Processing Operator Language (IPOL). This description language contain all needed components hardware components like Sensors, Displays, execution units and software parts like image processing algorithms.**

***Keywords*—DSL, design flow, image processing**

I. INTRODUCTION

Setting up an embedded application, which uses high performance image processing architectures is a very complex task. In the traditional industrial image processing field, engineers follow Moore's Law and use standard CPUs for their image processing applications. This solution is not resource and energy aware and therefore, does not work for embedded applications. Due to continuous rising requirements on the one hand, and physical limitations of embedded applications concerning area, time and energy, embedded image processing systems become more heterogeneous for fulfilling their functions. For example, in [1] [2] already heterogeneous systems consisting of FPGA, CPU and GPU were used for fast image processing. In general, the usage of an oversized general purpose hardware architecture is not allowed for fast embedded image processing. That leads to the approach of using more application-specialized computing architectures like GPUs or own specialized circuits in FPGAs (Field-Programmable-Gate-Arrays) or ASICs (Application-Specific-Integrated-Circuit).

Although, heterogeneous hardware is available, choosing the right parts (processors or computational units) and programming it, is a hard challenge. Every architecture uses its own language and follows its own programming paradigm. Examples are simple processors (e.g., ARM) with C/C++, FPGAs with VHDL and Verilog, GPUs with CUDA. Moreover, a written, hand crafted solution for one architecture, does not allow the port to another architecture. Therefore, more abstract (and parallel) languages have been developed in the past, which allows an automatic compilation for different architectures. One examples is OpenCL which supports GPUs as well as CPUs. Also a backend for Altera and Xilinx FPGAs were added. But these languages are used for general purpose and they are not specialized for a specific domain, e.g. image processing. Moreover, they support just one target at compile time, which means the code is then executed either at a GPU or FPGA or CPU - they do not work simultaneously together on a problem. Other examples are existing high-level synthesis tools such as Vivado HLS [3] and Intel CoFluent [4]. These tools are often suboptimal and not fully developed. The bad performance of this tools results from the non-specific approach. The tools are too general and do not consider the advantages of image processing architectures. In our design flow we are focused on the image processing domain and use image specialized hardware architectures such as the Full Buffering [5].

Thus, the design flow of mapping complex image processing applications on heterogeneous architectures is a tough challenge and not sufficiently solved in the past. Complex image processing algorithms with different tasks in different granularity have different hardware requirements. These algorithms have to utilize the advantages of specialized hardware architectures for fulfilling the system constraints. Therefore, not only software engineers, but especially hardware engineers, application engineers and system designers are needed, in order to cover all parts of such a system development. Regular programming languages like C/C++ or hardware description languages like VHDL are not applicable for that, because they do not promote a holistic view of the system development. These languages lead to a strict separation of software and hardware and do not consider evaluation techniques for the whole system. With the IPOL language, we want to introduce a holistic domain specific system description language for hardware-software co-design.

This paper is organized as follows. The next section presents the IPOL itself. After that, we present a workflow, how this language can help to find a suitable hardware architecture for a given image processing algorithm. In Section III,

 ISBN: 978-1-61208-399-5

an example based on a simulation environment is presented. At the end, we give a conclusion and outlook for future work.

II. Image Processing Operator Language (IPOL)

As described above, with IPOL a whole image processing pipeline could be modeled. Therefore, a detailed description of such a pipeline has to be shown which can then transfered to the language. Normally, an image is acquired using a camera consisting an image sensor. Due to different types of sensors, the data stream from the sensors can vary. Some parameters consist of resolution, bit per pixel, count of pixels per clock cycle, etc. After that, the image has to be processed. This is done via a concatenation of different operators. For example, Gauss, Median and Sobel, can also be parametrized e.g., with the size of the filter mask. Moreover, complex operators like Hough [6] or Fast-Fourier-Transformation have to be executed. After all operators processed the image, an output device has to be specified. That could be for example an display with a parameterizable resolution.

With IPOL it is possible to model such an image processing pipeline in a formal way. Therefore, IPOL is a XML based language. In that language components of an image processing, as described above, exist as a XML-component. In general there are existing 3 types: Sources (e.g. Sensor), Processing (e.g. Operators) and Sinks (e.g. Display). All these components can be extended with content-related parameters (e.g. filter-mask for gauss). In Figure 1 the structure of IPOL is demonstrated.

The example shows the whole image processing application named ”operatorchain”, with a sensor, display and two image processing operators (sobel filter and hough transformation). In this example, both image processing operators work on each pixel in the image. Other possibilities could be for example partitioning, where a defined area of pixels is reduced to a feature. The properties ”input_area” and ”output_area” specify the memory access pattern of every operator. An operator with a discrete access window has an other access pattern as an operator with a random access of the whole image. The ”base_calc” block includes a formal description of the algorithm. In this paper ”base_calc” remain omitted. In the example of Figure 1 the image processing operator named Sobel needs a 3×3 neighborhood of picture elements for the calculation of an 1×1 output region. The input and output area of each operator could vary. As shown in Figure 1 an other image processing algorithm, e.g. the Hough Transformation [7], has a different input and output behavior as the Sobel operator. In that example the Hough operator needs only one pixel for a processing step. The ”input_area” of the Sobel is larger, but with a smaller ”output_area” than the Hough operator. For unknown image processing algorithms, our approach provides a library of common image processing operations, like matrix operations. These library could be used for creating new custom image processing algorithms. The known behavior of the common image processing operations makes it possible to analyze these kind of algorithm without knowing the algorithm itself. If an image processing system is specified in such a language, an automatic optimization and derivation to hard- and software components becomes possible. This approach was often discussed in hardware-software co-design publications [8] [9], but investigated only small grained architectures with static solutions and not specialized

```
<operatorchain>
<globalconstraints>
        <accuracy>20</accuracy>
        <powerconsumption>20</powerconsumption>
        <fps>20</fps>
        ...
</globalconstraints>

<component id="0">
  <type>Sensor</type>
  <res><x>1920</x><y>1080</y></res>
  <pixres>12</pixres>
  <fps>30</fps>
</component>

<component id="1">
        <type>Operator</type>
        <access>each_pixel</access>
        <name>Sobel</name>
        <input_area><x>3</x><y>3</y></input_area>
        <output_area><x>1</x><y>1</y></output_area>
        <base_calc><src>...</src></base_calc>
</component>

<component id="2">
        <type>Operator</type>
        <access>each_pixel</access>
        <name>Hough</name>
        <input_area><x>1</x><y>1</y></input_area>
        <output_area><x>3000</x><y>3000</y></output_area>
        <base_calc><src>...</src></base_calc>
</component>

<component id="3">
  <type>Display</type>
  <res><x>800</x><y>600</y></res>
  <pixres>12</pixres>
  <fps>60</fps>
</component>

<connections>
  <con><out>0</out><in>1</in></con>
  <con><out>1</out><in>2</in></con>
  <con><out>2</out><in>3</in></con>
</connections>
</operatorchain>
```

Fig. 1. Example of an operator chain in the image processing language

for image processing applications. In this paper, we propose an additional approach for whole image processing systems, with heterogeneous architectures consisting of complete processing units like CPU cores, GPUs or special architectures on FPGAs. Detailed apsects will be discussed in Section III.

III. System Optimization

With IPOL it is feasible to introduce a new system design workflow for image processing systems. The design flow will cover all the aspects of system design, to consider non-functional properties in the abstract design layers. Therefore, a description of an image processing system utilizing IPOL allows an abstract and programming language independent development. However it is bound to the image processing domain. The new structural features provide the basis for later mapping into concrete hardware and tracing back hardware features to the UML. Figure 2 shows the concept of an image processing design flow.

 ISBN: 978-1-61208-399-5

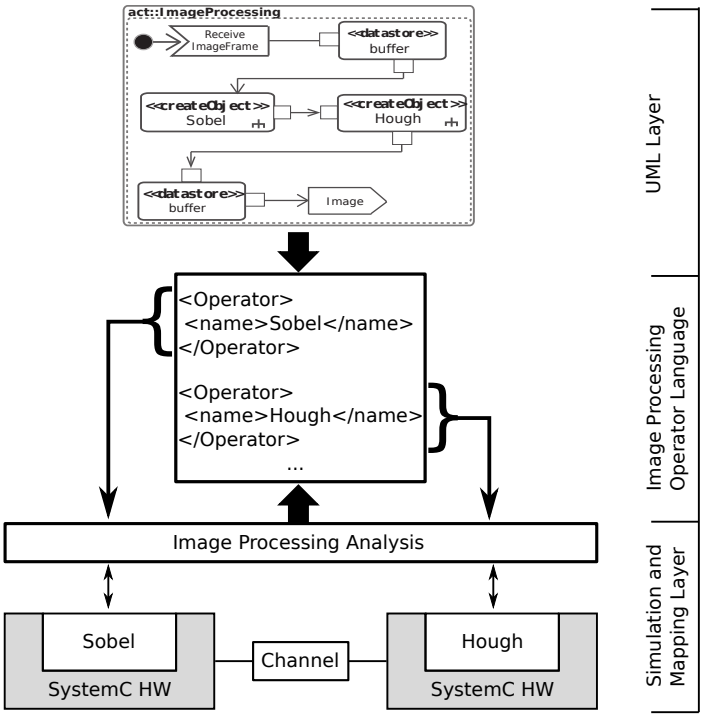

Fig. 2. Image processing design flow: The design flow enables the possibility to create an image processing application using UML.

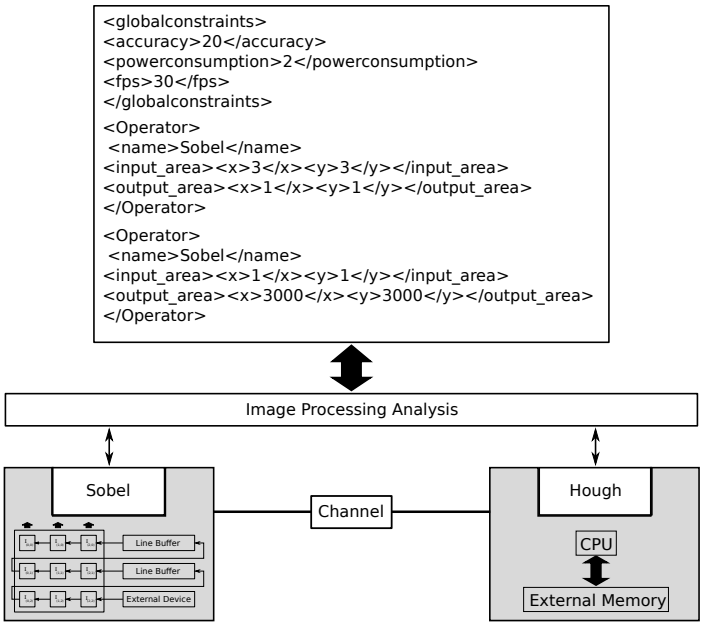

Fig. 3. Image processing analysis with an image processing language

It enables the user to develop the image processing application in an abstract layer such as UML. An automated mapping of the image processing operators to the simulation environment with virtual hardware makes a holistic UML-based design approach feasible. Thus the user is able to design an image processing application without detailed knowledge of the underlying hardware architecture. The UML model will be automatically transfered in an executable SystemC simulation. This is done by the rules of the domain-specific language, image processing operator language (IPOL) and controlled by the image processing analysis. A more detailed view on the optimization tool is shown at Figure 3. In contrast to existing approaches the image processing analysis considers the in- and output pattern of the algorithms for an automated mapping on specialized or general hardware.

By the example of Figure 3 the Sobel algorithm has a 3×3 input and a 1×1 output environment. The second algorithm, called Hough has a different data access pattern. This algorithm needs only one input pixel for processing, but a 3000×3000 area for the output. Such memory access pattern influences the image processing analysis regarding the hardware architecture selection. In the example of Figure 3 the Sobel is mapped on the full buffering architecture [1]. Specialized hardware such as full buffering could be used for algorithms with strong limited input and output environments. An algorithm with a random or widely scattered data access pattern is not suitable for that kind of architecture. The Hough algorithm would be mapped on an other architecture, with a good connection to external memory. Because of its large amount of memory in the output area. The memory access pattern would be checked for all algorithms in the image processing application. Additionally to the algorithm behaviour, domain specific requirements could be defined, shown at Figure 3. They are called non-functional properties or "globalconstraints". The "globalconstraints" are used as inputs for the system optimization. The image processing analysis uses the non-functional properties to find a suitable architecture for fulfilling the constraints by using the least resources. In the example of Figure 3 the image processing analysis reads the system requirements : $accurarcy = 20$, $powerconsumption = 2$ and $fps = 30$ and proposes a system architcture. This means if the image processing system makes 20 frames per second (fps) with the configuration of one, 40 fps with two and 60 fps with three full buffering processing elements. The system will select two processing elements for fulfilling the requirements of $fps = 30$. A graphical representation of the optimization example is shown at Figure 4.

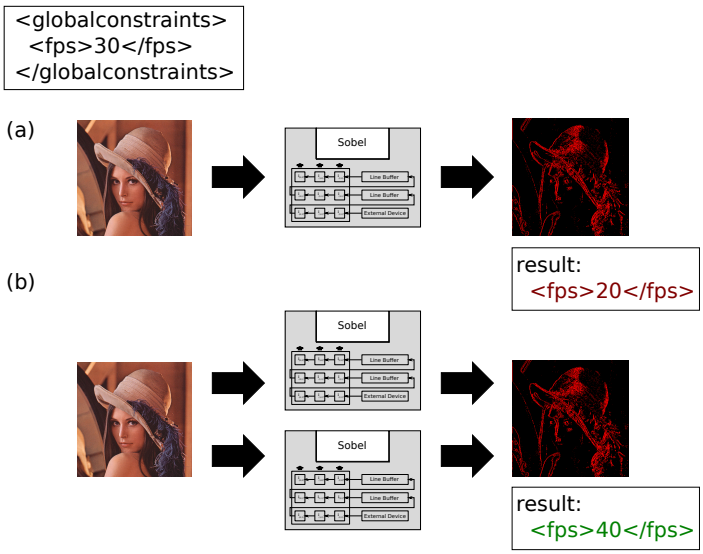

Fig. 4. Image processing analysis: Global constraints and hardware mapping

 ISBN: 978-1-61208-399-5

It shows that the architecture with one processing element do not perform the 30 fps. An architecture with two processing elements passes the test. This optimization will be done for all algorithms and all "globalconstraints". Once the static mapping is done the dynamic SystemC simulation run the image processing application with the selected hardware for testing the system configuration. Depending on the simulation result, the system will be mapped on real hardware or has to make an other iteration of the optimization. By a new optimization step the simulation results serve as input for the image processing analysis.

IV. Conclusion and Future Work

In this paper, a new concept and first steps of the underlying methodology have been introduced for modeling and implementing complex image processing systems with a holistic top down approach on heterogeneous computer architectures. The approach covers all layers from abstract UML to a domain-specific language to the executable specification with virtual hardware down to real hardware. In one of our next research steps, we are going to design a connection to the Open Virtual Platform (OVP) [10] and SoClib [11]. That will help developers to include the simulation results for specific processor architectures in their model, without the need of possessing that processor in physical.

Acknowledgment

This work was financially supported by the Research Training Group 1773 "Heterogeneous Image Systems", funded by the German Research Foundation (DFG).

References

[1] M. Schmidt, M. Reichenbach, and D. Fey, "A Smart Camera Processing Pipeline for Image Applications Utilizing Marching Pixels," in *Signal and Image Processing : An International Journal (SIPIJ)*. SIPIJ, 2011, pp. 137–156.

[2] M. Reichenbach, R. Seidler, and D. Fey, "Heterogeneous Computer Architectures: An Image Processing Pipeline for Optical Metrology," in *Proceedings of ReConFig*. IEEE, 2012, pp. 1–8.

[3] "Vivado HLS," April 2015. [Online]. Available: http://xilinx.com

[4] "Intel Cofluent," April 2015. [Online]. Available: http://intel.com

[5] M. Schmidt, M. Reichenbach, and D. Fey, "A generic vhdl template for 2d stencil code applications on fpgas," in *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*. IEEE, October 2012, pp. 180–187.

[6] W. Burger and M. Burge, *Principles of Digital Image Processing*. Springer, 2009.

[7] J. C. Russ, *The Image Processing Handbook*. Crc Pr Inc, 2011.

[8] F. Mischkalla, D. He, and W. Mueller, "Closing the Gap between UML-based Modeling, Simulation and Synthesis of Combined HW/SW Systems," in *Design, Automation and Test in Europe (DATE)*, 2010.

[9] A. Fidjeland and W. Luk, "Archlog: High-Level Synthesis of Reconfigurable Multiprocessors for Logic Programming," in *Field Programmable Logic and Applications, 2006. FPL '06*. IEEE, 2011, pp. 1–6.

[10] "Open Virtual Platform," April 2015. [Online]. Available: http://www.ovpworld.org/

[11] "SoClib," April 2015. [Online]. Available: http://soclib.fr

 ISBN: 978-1-61208-399-5