Evaluating OpenCL as a Standard Hardware Abstraction for a Model-based Synthesis Framework: A Case Study

Omair Rafique and Klaus Schneider
Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany

Keywords: Model-based Design, Heterogeneous Synthesis, Parallel Computing Languages.

Abstract: In general, model-based design flows start from hardware-agnostic models and finally generate code based on the used model of computation (MoC). The generated code is then manually mapped with an additional non-trivial deployment step onto the chosen target architecture. This additional manual step can break all correctness-by-construction guarantees of the used model-based design, in particular, if the chosen architecture employs a different MoC than the one used in the model. To automatically bridge this gap, we envisage a holistic model-based design framework for heterogeneous synthesis that allows the modeling of a system using a combination of different MoCs. Second, it integrates the standard hardware abstractions using the *Open Computing Language (OpenCL)* to promote the use of vendor-neutral heterogeneous architectures. Altogether, we envision an automatic synthesis that maps models using a combination of different MoCs on heterogeneous hardware architectures. This paper evaluates the feasibility of incorporating OpenCL as a standard hardware abstraction for such a framework. The evaluation is presented as a case study to map a synchronous application on different target architectures using the *OpenCL specification*.

1 INTRODUCTION

A heterogeneous embedded system consists of different devices including possibly single-core and multi-core processors with application-specific hardware and even more specific sensors and actors. At the level of its software architecture, it may consist of many components concurrently running on these devices that interact with each other via particular models of computation (MoCs). To develop such complex embedded systems, a new modeling and programming paradigm has been introduced by model-based design: A model-based design is generally characterized with a hardware-agnostic abstract model and is supported by a complete tool chain typically providing simulators, tools for verification, code generators, and tools for system and communication synthesis.

Apart from modeling tools, many specification languages (Dagum and Menon, 1998; Stone et al., 2010) have been introduced to target high-performance computing in highly parallel and heterogeneous architectures. Among them, the Open Computing Language (OpenCL) has gained a tremendous amount of popularity and support by the leading hardware vendors including Intel, Apple, AMD and many others. In contrast to proprietary specification languages with limited hardware choices, OpenCL allows task-parallel and data-parallel heterogeneous computing on a variety of modern CPUs, GPUs, DSPs, and other microprocessor designs (Stone et al., 2010).

The existing model-based design methodologies are based on different models of computation (MoCs) (Lee and Messerschmitt, 1987; Benveniste et al., 2003; Cassandras and Lafortune, 2008). We appreciate the convenient use of model-based design frameworks, but we also address one of the major limitations of their application: The final result of these design flows is essentially a set of automatically generated C files that have to be deployed in an additional non-trivial manual step to a particular target architecture. We termed this gap as the *mapping/deployment gap*. To further automate the design process and to bridge this gap, we envisage a holistic model-based design framework that allows modeling behaviors with a heterogeneous combination of MoCs, and employs the OpenCL specification to abstract from the real hardware platforms. This way, we can automatically synthesize the modeled behavior on any OpenCL-abstracted targeted hardware thereby bridging the aforementioned mapping/deployment gap.

This paper presents a preliminary case study to an-

Rafique, O. and Schneider, K.
Evaluating OpenCL as a Standard Hardware Abstraction for a Model-based Synthesis Framework: A Case Study.
DOI: 10.5220/0007470503860393
In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019)*, pages 386-393
ISBN: 978-989-758-358-2

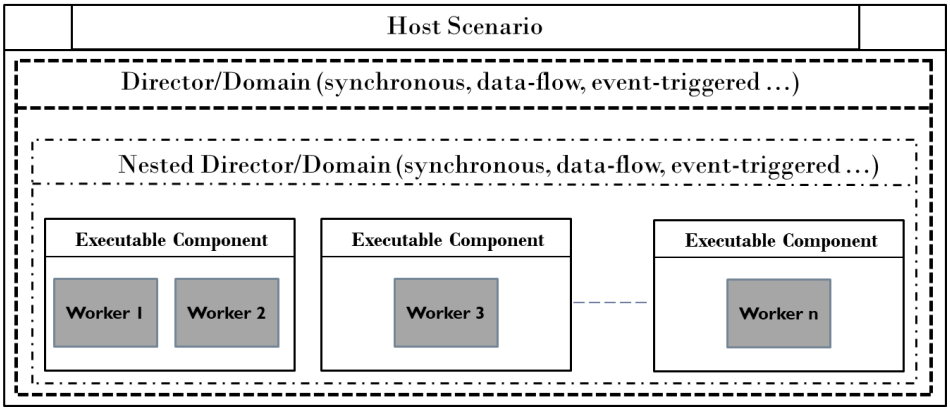


Figure 1: The basic building block diagram of the concept.

alyze the feasibility of OpenCL as a standard hardware abstraction for the use in a model-based design framework for the fully automated system synthesis. To this end, the case study based on a computation-intensive synchronous application is implemented using the proposed framework and the results are evaluated against two main features offered by OpenCL, i.e., the cross-vendor portability and the substantial performance acceleration in parallel architectures.

2 THE FRAMEWORK

A holistic model-based design framework for heterogeneous synthesis is conceived as a completely automatic design flow from a specification to a target hardware architecture. For that reason, the framework is designed to provide and practice two primary concepts: First, the concept of modeling a system with various MoCs (Lee and Messerschmitt, 1987; Benveniste et al., 2003; Cassandras and Lafortune, 2008) or even with a heterogeneous combination of these MoCs (Eker et al., 2003; Kuhn et al., 2013), and second, the concept of using vendor-neutral heterogeneous architectures as an integral part of the framework, as realized, e.g., by means of standard abstractions like OpenCL.

2.1 Heterogeneous Modeling

Model-based design flows are based on models of computation (MoC) that precisely determine *why, when and which atomic action of a system is executed*. Thus, a MoC specifies in general what triggers the execution of a component and how these components communicate with each other. Model-based design frameworks like Ptolemy II (Brooks et al., 2010) and FERAL (Kuhn et al., 2013) also support modeling a system based on a heterogeneous combination of MoCs. These frameworks provide a common platform for organizing a system into different domains characterized as directors. Each enclosing director represents a semantic model based on a specific MoC and triggers the execution of the contained components in accordance to the implemented semantics. The heterogeneous combination of MoCs is therefore realized by coupling different directors within an application scenario. The proposed framework therefore adopts and extends the concepts of these frameworks for synthesizing a heterogeneous combination of MoCs to real heterogeneous architectures.

2.2 OpenCL Abstraction

OpenCL is an open specification language designed for heterogeneous parallel computing on cross-vendor and heterogeneous architectures. The basic objective of OpenCL can be understood from two primary benefits it offers: First, OpenCL provides an abstract platform model that can be exploited for substantial acceleration in parallel computing. To this end, it supports both coarse-grained (task-level) as well as fine-grained (data-level) parallelism. Second, it provides the ability to write vendor-neutral cross platform applications. This is achieved as it provides upper-level abstractions hiding the lower-layer implementations (drivers and the runtime) as well as consistent memory and execution models to allow cross-vendor development. The basic strength of this abstraction is the ability to scale code from simple embedded microcontrollers to multi-core CPUs, up to highly-parallel GPU architectures, without revising the code.

Since a general discussion of OpenCL is out of scope for this paper, we refer to (Lee et al., 2015; Shen et al., 2013) for further details. For the proposed framework, the basic idea is to exploit the ability of OpenCL to write vendor-neutral cross platform applications.

2.3 Heterogeneous Synthesis

The idea to amalgamate the concept of heterogeneous modeling and the OpenCL abstraction under the supervision of a common framework leads to heterogeneous synthesis. For heterogeneous modeling, the proposed framework realizes a hierarchical structure to compose different models as realized in (Eker et al., 2003; Kuhn et al., 2013). With this approach, a complex system can be effectively modeled into a tree of nested subsystems as shown in Figure 1. Each node (a subsystem) of the tree may be composed of executable components, enclosed by a director/domain that enforces the execution and communication of the node based on the implemented semantics. These implemented semantics actually represent the MoC. The executable components are further composed of workers where the actual behavior of the subsystem is implemented. In other words, a worker is simply a C function that implements a part of the algorithm, and is triggered based on the MoC of the component. This complete hierarchy from directors to components, and up to the workers provides a common platform for modeling a complex system with heterogeneous combination of MoCs.

Apart from this openness and abstraction provided by the framework, OpenCL also does not impose a specific MoC. Instead, it offers an abstract and highly diffusive execution model that provides a transparent way of distributing an application for the acceleration of parallel computing. The OpenCL execution model comprises two components: the host program and kernels. A host is connected to one or more compute devices (CPU, GPU, DSP etc.) and is responsible for managing resources on compute devices, including the organization of the executions of kernel instances. Kernels are C-like functions that actually implement the abstract behavior of the system or part of the system. To this end, as shown in Figure 1, the OpenCL host as a main application scenario, can be modeled with the composition of directors/domains, whereas the kernels can be naturally integrated as executable components of the used domains. Each atomic instance of the kernel then represents a worker, capable of executing in parallel, a part of the complete behavior.

3 THE CASE STUDY

In this section, we present a complete case study based on a computation-intensive synchronous application. The basic application used as a test case is first discussed, followed by the discussion on how the application is modeled using the proposed framework, and finally we report about detailed evaluations to analyze various mappings of the application using the OpenCL execution model.

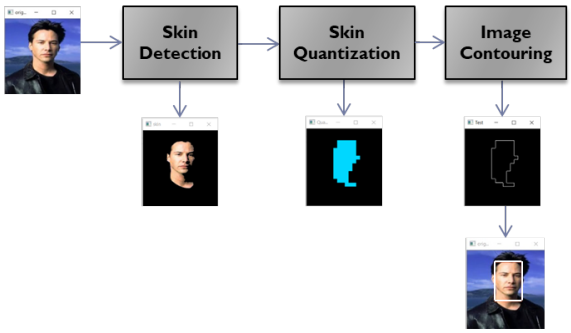


Figure 2: Face detection using minimal facial features.

3.1 The Synchronous Application

Instead of utilizing the available benchmarks that are already presented in other publications (Lee et al., 2015; Shen et al., 2013; Grewe and O'Boyle, 2011) for generally evaluating the OpenCL specification, we decided to develop a 2D-image processing algorithm namely, *the face detection using minimal facial features*. Image processing algorithms naturally allow a wide margin for parallel computing, and thus became a natural choice for evaluating the OpenCL model.

The algorithm is composed of three different modules as shown in Figure 2, and is based on the work presented in (Chen and Lin, 2007) where it was originally implemented in OpenCV[1]. Each module was transformed and implemented as an OpenCL kernel, where each module performs a specific processing task on the image, and runs synchronously with the preceding module. A brief discussion on each module is given in the following.

3.1.1 Skin Detection

The skin detection module performs a pixel-by-pixel processing of the image to decide whether a specific pixel lies in the defined range of the skin color. In contrast to (Chen and Lin, 2007), the range of the skin color is determined by the combination of the RGB model as well the normalized RGB model. Therefore, based on this range, this module simply tests each pixel and replaces non-skin color pixels with black ones. However, the final outcome (as shown in Figure 2) also replaces some of the skin color pixels with black ones. This is then sorted out in the next stage by the *skin quantization* module.

[1] http://docs.opencv.org

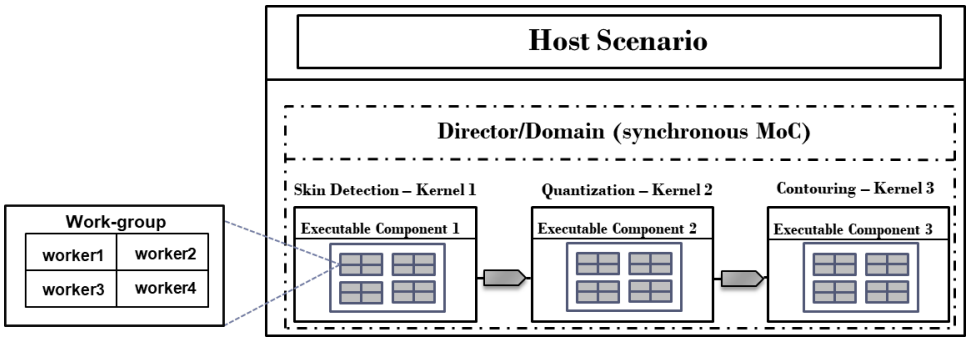


Figure 3: Application modeled under synchronous dataflow domain.

3.1.2 Skin Quantization

This module lowers the image resolution by splitting the whole image into blocks of 10x10 pixels. The number of black pixels within a block of hundred pixels are counted and if the total count is less than a specified threshold, the block is extracted as a part of the skin. As an example, for a 200x200 resolution image, the module splits it into four-hundred blocks, where each block contains 100 pixels. Each block is then evaluated with the specified threshold. This finally contributes in filtering out wrongly processed black pixels that are placed by the first module.

3.1.3 Image Contouring

Image contouring or contour tracing is a common technique applied on 2D images to extract information about their general shape. Hence, it can simply be applied to find out the boundaries of a digital image. A variety of contour tracing algorithms are available with their respective pros and cons (Seo et al., 2016). However, the image contouring module of the proposed algorithm specifically implements the *Moore-Neighbor tracing algorithm* to extract the boundaries of the quantized image, as shown in Figure 2.

3.2 Application Model using the Framework

The design model of the application using the proposed framework is depicted in Figure 3. As discussed, each module of the proposed algorithm implemented as an OpenCL kernel is realized and designated as an executable component of the framework. Each executable component denotes an abstract representation of the module behavior. The host program then determines the fine-grained parallelism where the abstract executable component is defined in a set of threads, termed as workers/work items to implement the complete behavior. These workers represent the execution instances of the associated kernel and can be grouped together in different work groups. Ideally, each work group is assigned to a single compute unit where a compute unit can simply be conceived as for e.g., a logical core of a CPU or a streaming multiprocessor of a GPU. To ensure consistency with the framework terminologies, we will use the terms executable components and workers for OpenCL kernels and work items respectively.

For this case study, the host provides a synchronous dataflow domain for the execution of the proposed algorithm. To this end, the algorithm composed of three different modules is scheduled based on a synchronous dataflow (SDF) MoC as shown in Figure 3. Based on the SDF MoC, the order in which modules are executed is specified statically at compile time. Practically, the host program based on the SDF MoC, enqueues the executable components on the same OpenCL command queue in chronological order. The executable components are triggered in sequence for execution and the final resulting image is then retrieved by the host.

3.3 Evaluation

The case study is dedicated to observe and analyze the two main claims associated with OpenCL namely, the cross-vendor portability and the substantial acceleration in parallel computing.

3.3.1 Cross-vendor Portabilty

The proposed framework is envisioned to support the implementation of vendor-neutral heterogeneous architectures. To this end, it is important to evaluate and to collect preliminary results as a baseline to employ OpenCL for implementing vendor-neutral heterogeneous architectures within the framework. Therefore, this case study explores the ability of OpenCL to facilitate cross-vendor portability by employing devices of different types and from different vendors. The already discussed algorithm implemented

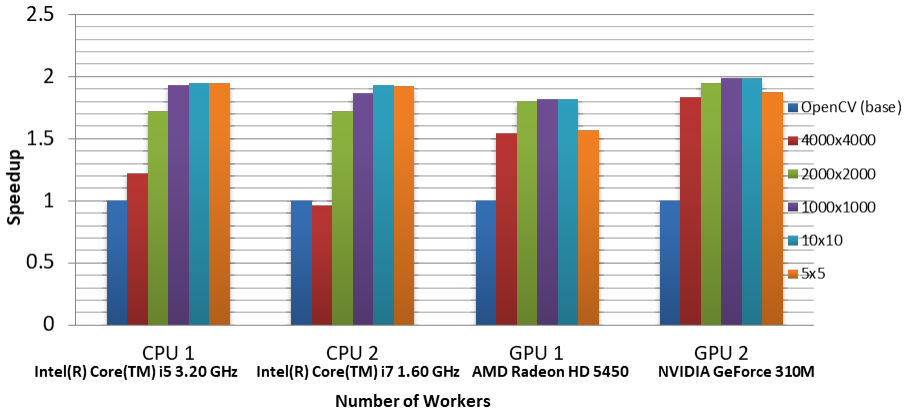


Figure 4: Workers vs Speedup.

with the OpenCL specification is evaluated on different devices including different CPUs and GPUs from three different vendors, namely *Intel*[2], *AMD*[3] and *NVIDIA*[4].

3.3.2 Parallel Computation using OpenCL

The OpenCL model offers a very abstract environment for distributing and implementing an algorithm with different levels of parallelism. The proposed framework exploits this abstract execution model provided by OpenCL to better utilize the available hardware resources and substantially accelerate the parallel computations. To this end, organizing and distributing executable components with different number of workers and work groups to express and to evaluate various levels of parallelism is considered.

An executable component describes the behavior of a single worker, and the host program then explicitly declares the number of workers to decide the parallelism of the application. The total number of workers represent the global work size of the component and can further be arranged and divided in work groups or compute units. The number of workers per work group represents the local work size. To this end, it is important to tune these parameters (workers and work groups) to find the optimal level of parallelism required for a particular architecture.

3.3.3 Results

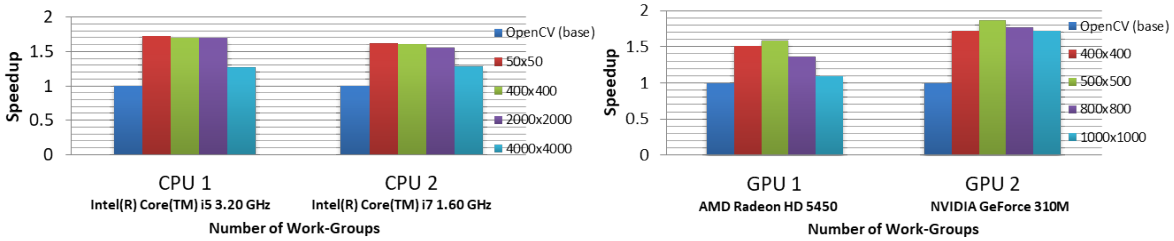
We have measured the speedup factor against the total number of workers and work groups, respectively. The speedup is basically calculated with reference to the computation time (in seconds) of the OpenCV-based naive implementation. The computation time actually represents the total time taken by the complete algorithm to finally detect the face in the given input image.

Number of Workers. We introduced an additional parameter namely, the *block-size* that allows us to manage the amount of workload associated with each worker. The block size actually specifies the number of pixels that will be processed by each worker. Consequently, increasing the block size implies increasing the workload per worker and hence decreasing the total number of workers.

As shown in Figure 4, the speedup factor is measured for different global sizes ranging from the highest possible parallel implementation where the number of workers is equal to the total number of pixels (4000x4000), up to the least number of just 25 (5x5) workers. Decreasing the workers implies an increase in the block size, where more workload is then managed by each worker. The number of workers and the block size affect performance differently on CPUs and GPUs, based on different architectural characteristics. Since GPUs generally accommodate a large number of streaming cores/CUDA cores, they are capable of managing a large number of parallel threads. On the contrary, the ability of a CPU to handle parallel threads is just limited to the number of few available cores.

Based on the results, first, even with the API overhead, the OpenCL implementation clearly outperforms the original naive implementation with different number of workers on different devices. Second, on CPUs, we clearly observe a performance gain with larger block sizes, implying less number of workers. This is because executing a large number of workers on few available cores results in significant schedul-

[2]https://software.intel.com
[3]https://developer.amd.com/
[4]https://developer.nvidia.com



(a) CPUs (b) GPUs

Figure 5: Work-Groups vs. Speedup.

ing overhead on CPUs and thus reduces the overall performance. Also, we observe that increasing the block size to a certain level (after 1000x1000 workers) saturates the performance gain. This is because with each worker having sufficient workload results in reduced the scheduling overhead.

On the contrary, GPUs outperform CPUs with larger number of workers. However, with reduced number of workers, especially when the size gets lower than the available number of cores, the performance starts to degrade substantially. This is because the available cores are not occupied, and hence the processing capability is not completely utilized.

Number of Work Groups. The work group size specifies the number of workers in a group. On GPUs, a work group or multiple work groups are executed on a streaming multiprocessor (SM) where ideally each worker is allocated to a streaming core/SIMD lane. On CPUs, a work group is allocated to a logical core. Since the employed CPUs and GPUs have a different work group sizes limit, two separate plots are presented, one for each type of device, as shown in Figure 5.

Similar to the case of workers, even with different work group sizes, the OpenCL implementation outperforms the original naive implementation. As shown in Figure 5a, on CPUs with an increasing number of work groups (decreasing work group size) the performance decreases. With the highest number of work groups, i.e., by 4000x4000 work groups, where each work group is composed of a single worker, the performance degrades substantially. This is because with a larger number of work groups, the scheduling overhead increases on CPUs.

As shown in Figure 5b, similar to CPUs, the performance also decreases with increasing number of work groups on GPUs. This is because with a larger number of work groups, the work group size decreases, and with a smaller number of workers per work group, the processing units of the SM are not completely utilized. This effect can be clearly observed with the highest number of work groups used as in the case of 1000x1000 work groups.

Hence, the presented evaluations allow us to collect preliminary results that can be used as a baseline by the proposed framework to efficiently map models based on the available resources as provided by the target hardware. Also, it can be derived that OpenCL can be employed as a standard abstraction for the proposed framework to implement vendor-neutral heterogeneous architectures.

4 RELATED WORK

The related work is observed from two main aspects as given in the following sections.

4.1 Model-based Design Frameworks without OpenCL

Model-based design methodologies in the related state-of-the-art mainly differ by their employed MoCs. In (Bezati et al., 2014), the HW/SW co-design methodology based on the CAL actor programming language is built as an Eclipse plug-in on top of ORCC[5] and OpenForge[6]. The open-source tools are used as a tool chain of the framework, capable of providing simulation and the HW/SW synthesis. The methodology also divulges design space exploration techniques. A similar approach namely System-CoDesigner is presented in (Haubelt et al., 2008) by the University of Erlangen-Nuremberg.

The most popular and commercially recognized model-based design tool Matlab[7] has introduced a variety of supporting toolkits over time. The modeling toolkit *Simulink* provides a graphical extension for modeling and simulation of systems. Similarly, the *Embedded Coder* generates C and C++ files for embedded software processors. The *Simulink Design*

[5]http://orcc.sourceforge.net
[6]https://sourceforge.net/projects/openforge
[7]http://www.mathworks.com/matlabcentral/

Verifier and *Polyspace* are introduced for the formal verification of models and code, respectively. However, an interesting approach is presented in (Stefanov et al., 2004) where the Matlab code is transformed to a KPN specification using the Compaan compiler. The HW backend Laura is used to map this KPN specification to hardware.

Apart from these frameworks that support homogeneous modeling, Ptolemy (Eker et al., 2003) and Ptolemy II (Brooks et al., 2010) support multiple MoCs including dataflow process networks, discrete-event models, synchronous/reactive models and many more. These frameworks employ a hierarchical structure to compose different models under the supervision of software components called *directors*. Directors control the semantics of the execution of components (actors) as well as the communication between actors. Consequently, this allows modeling a system with heterogeneous combination of MoCs.

FERAL is another framework that allows heterogeneous modeling and simulation (Kuhn et al., 2013). It is developed to provide a holistic model-based design approach to enable the coupling of specialized simulators in offline scenarios, i.e., without connecting them to real hardware. This project very interestingly adopts and extends some of the concepts from the Ptolemy project. The proposed framework adopts and further extends some of the concepts from FERAL and Ptolemy in order to support the cause of modeling behaviors with heterogeneous combination of MoCs.

4.2 Model-based Design Frameworks with OpenCL

Some of the existing model-based design frameworks like those presented in (Boutellier and Hautala, 2016; Lund et al., 2015; Schor et al., 2013) employ OpenCL for better exploiting the parallelism offered by heterogeneous architectures. To this end, the framework presented in (Schor et al., 2013) introduces a design flow for executing applications specified as synchronous dataflow (SDF) graphs on heterogeneous systems using OpenCL. The main focus of this work is to develop and to provide features and concepts to better utilize the parallelism and thereby improving end-to-end throughput in heterogeneous architectures.

Another similar approach presented in (Boutellier and Hautala, 2016) is aimed to provide a dataflow programming framework not restricted to the SDF MoC only. Hence, the framework targets modeling a system based on dynamic dataflow and allows the mapping of actors with a data-dependent consumption of inputs and a data-dependent production of outputs.

Similarly, (Lund et al., 2015) introduces a translation methodology for translating dataflow process networks (DPNs) into programs running some of the computations on the OpenCL platform. Consequently, it allows the mapping of DPN networks described in RVC-CAL to a data parallel architecture consistent with the OpenCL API.

However, all the cited frameworks focus only on mapping a system modeled with the dataflow MoC to heterogeneous architectures using OpenCL. Consequently, the ability to map heterogeneous models to heterogeneous architectures under the supervision of a common framework is still desired.

5 CONCLUSION AND FUTURE WORK

In this paper, we first introduced the proposed framework that enables the modeling of systems with a heterogeneous combination of MoCs using the concept of hierarchical composition of semantic domains. Moreover, the concept of implementing portable heterogeneous architectures is realized by introducing OpenCL as a standard hardware abstraction for the model-based design of embedded systems. In this paper, we presented a preliminary case study to evaluate the use of OpenCL as an abstract hardware architecture. For this reason, an application modeled by a synchronous dataflow MoC was developed, deployed, and evaluated using various parameter settings provided by OpenCL. The results obtained let us conclude that OpenCL provides a very abstract environment for implementing applications at different levels of parallelism and thus became a natural choice for facilitating the cause of automatic synthesis in the framework. Based on this preliminary case study, future work is aimed to use OpenCL as a standard hardware abstraction to synthesize systems modeled with a heterogeneous combination of MoCs for a heterogeneous target hardware platform.

REFERENCES

Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Le Guernic, P., and de Simone, R. (2003). The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1):64–83.

Bezati, E., Thavot, R., Roquier, G., and Mattavelli, M. (2014). High-level dataflow design of signal processing systems for reconfigurable and multicore heterogeneous platforms. *Journal of Real-Time Image Processing*, 9(1):251–262.

Boutellier, J. and Hautala, I. (2016). Executing dynamic data rate actor networks on OpenCL platforms. In *Signal Processing Systems (SiPS)*, pages 98–103, Dallas, TX, USA. IEEE Computer Society.

Brooks, C., Lee, E., and Tripakis, S. (2010). Exploring models of computation with Ptolemy II. In Givargis, T. and Donlin, A., editors, *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 331–332, Scottsdale, Arizona, USA. ACM.

Cassandras, C. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer, 2 edition.

Chen, Y.-J. and Lin, Y.-C. (2007). Simple face-detection algorithm based on minimum facial features. In *IEEE Industrial Electronics Society (IECON)*, pages 455–460, Taipei, Taiwan. IEEE Computer Society.

Dagum, L. and Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55.

Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y. (2003). Taming heterogeneity – the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144.

Grewe, D. and O'Boyle, M. (2011). A static task partitioning approach for heterogeneous systems using OpenCL. In Knoop, J., editor, *Compiler Construction (CC)*, volume 6601 of *LNCS*, pages 286–305, Saarbrücken, Germany. Springer.

Haubelt, C., Schlichter, T., Keinert, J., and Meredith, M. (2008). SystemCoDesigner: automatic design space exploration and rapid prototyping from behavioral models. In Fix, L., editor, *Design Automation Conference (DAC)*, pages 580–585, Anaheim, California, USA. ACM.

Kuhn, T., Forster, T., Braun, T., and Gotzhein, R. (2013). FERAL - framework for simulator coupling on requirements and architecture level. In *Formal Methods and Models for Codesign (MEMOCODE)*, pages 11–22, Portland, OR, USA. IEEE Computer Society.

Lee, E. and Messerschmitt, D. (1987). Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245.

Lee, J., Nigania, N., Kim, H., Patel, K., and Kim, H. (2015). OpenCL performance evaluation on modern multicore CPUs. *Scientific Programming*, pages 859491:1–859491:20.

Lund, W., Kanur, S., Ersfolk, J., Tsiopoulos, L., Lilius, J., Haldin, J., and Falk, U. (2015). Execution of dataflow process networks on OpenCL platforms. In *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 618–625, Turku, Finland. IEEE Computer Society.

Schor, L., Tretter, A., Scherer, T., and Thiele, L. (2013). Exploiting the parallelism of heterogeneous systems using dataflow graphs on top of OpenCL. In *IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia)*, pages 41–50. IEEE Computer Society.

Seo, J., Chae, S., Shim, J., Kim, D.-C., Cheong, C., and Han, T.-D. (2016). Fast contour-tracing algorithm based on a pixel-following method for image sensors. *Sensors*, 16(3):353:1–353:27.

Shen, J., Fang, J., Sips, H., and Varbanescu, A. (2013). An application-centric evaluation of OpenCL on multicore CPUs. *Parallel Computing*, 39(12):834–850.

Stefanov, T., Zissulescu, C., Turjan, A., Kienhuis, B., and Deprettere, E. (2004). System design using Kahn process networks: The Compaan/Laura approach. In *Design, Automation and Test in Europe (DATE)*, pages 340–345, Paris, France. IEEE Computer Society.

Stone, J., Gohara, D., and Shi, G. (2010). OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science and Engineering*, 12(3):66–73.