An OpenCL-based Framework for Rapid Virtual Prototyping of Heterogeneous Architectures

Efstathios Sotiriou-Xanthopoulos∗, Leonard Masing**, Kositas Siozios∗, George Economakos∗, Dimitrios Soudris∗ and Jürgen Becker∗∗
∗School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece
E-mail: {stasot, kksiop, geconom, dsoudris}@microlab.ntua.gr
**Institute for Information Processing, Karlsruhe Institute of Technology, Karlsruhe, Germany
E-mail: {leonard.masing, becker}@kit.edu

Abstract—The increasing performance and power requirements in embedded systems has lead to a variety of heterogeneous hardware architectures, featuring many different types of processing elements. This heterogeneity however induces extra effort on system development and programming. To address this heterogeneity, OpenCL provides a portable programming model which enables the use of one source code in various architectures featuring different types of processors. Also, such systems impose higher design complexity due to the existence of an increased number of hardware components. Virtual Prototyping aims to alleviate this issue by enabling the hardware modeling in higher abstraction levels. This paper combines the benefits of OpenCL with Virtual Prototyping, by proposing an OpenCL-based framework for rapid prototyping, which (a) automatically derives a virtual prototype from an OpenCL code; (b) executes the OpenCL application by running the host program along with the hardware simulation; and (c) proposes a design flow for faster system evaluation, as compared to state-of-the-art FPGA-based flow. Using a set of OpenCL benchmarks, it is shown that the proposed framework enables faster prototyping by up to 18x, as compared to state-of-the-art flow.

I. INTRODUCTION

Due to the ever increasing need for more processing power despite the limited energy budget available, efficient data processing is becoming more and more imperative. Thus, heterogeneous multi-processor Systems-on-Chip (MPSoC) have been an effective selection, as their hardware components can be customized to the exact application requirements. To exploit their full potential, choosing the right architecture for the running application is a requirement of utmost importance. This however imposes increased design effort for both software and hardware, especially when different types of processors (e.g. CPUs, GPUs, FPGAs etc.) are taken into consideration.

To solve the difficulties imposed by the programming heterogeneity of such platforms, OpenCL [1] provides a portable programming model which enables the programming of different types of processing elements, without the need for adapting the source code to each type. Hence, the designer is able to investigate multiple data processing architectures without extra programming effort. Although originally directed to GPU programming, the FPGA community is increasingly adopting OpenCL, thus enabling the easier and more efficient programming of FPGA devices.

However in the FPGA or ASIC world these nearly limitless customization options during the MPSoC design increase the design complexity. This is caused by the numerous architectural parameters in RTL design (e.g. when using FPGAs). Thus choosing an efficient architecture is a tedious and slow task; doing this task manually by experienced developers can take a lot of effort and man-month, making it unfeasible for most use cases. Virtual Prototyping has been proposed to alleviate this problem: The hardware is modeled in a software representation called Virtual Platform (VP), typically written in SystemC. The main benefit of such an approach is the hardware modeling in various abstraction levels, in each of which a number of architectural details is removed, thus limiting the architectural parameter combinations, especially in early design stages when some of the architectural details are not yet available. This enables the early software development and design space exploration, targeting to easier bug fixing, better design space coverage and shorter time-to-market. The VP might also serve as a golden reference to the development team.

The goal of this work is to propose a rapid Virtual Prototyping framework which (a) enables the automated VP modeling of heterogeneous hardware architectures by taking as input an OpenCL source code; (b) provides a platform-independent simulation environment between the hardware model and the host (i.e. the processor that coordinates the simulated hardware) without the need for real hardware platforms; and (c) is accompanied with a design flow that features faster development and evaluation cycles during a design space exploration procedure, as compared to state-of-the-art FPGA-based system design using OpenCL.

The paper is structured as follows: Sections II and III present the motivation and the related work of this paper. Section IV explains the proposed methodology. Section V shows the experimental results and discusses the insights gained. Finally, we conclude our work in Section VI.

II. MOTIVATION

To better clarify the benefits of combining OpenCL with Virtual Prototyping, we consider two scenarios, which are related to state-of-art design flows: (i) OpenCL without using VPs (e.g. using FPGA only) and (ii) VP modeling without using OpenCL (i.e. manual hardware modeling and CPU programming according to the hardware architecture).

OpenCL without VPs: The designer would use a FPGA-based platform for the system design and evaluation. This leads
to a vendor-dependent low-level RTL design with a specific supported architectural scheme, similar to Figure 1a, which depicts a typical bus-based SoC with one FPGA fabric and a dual-core ARM. However there might be alternative architectures to be taken into consideration during SoC design: For example, Figure 1b depicts a cluster of sub-systems, following the architectural pattern of Synopsys HAPS [2]: Each sub-system is similar to the SoC of Figure 1a and executes a different set of threads. Figure 1c depicts a SoC where the CPUs and memories are decoupled from the FPGAs, while Figure 1d shows a NoC-based system, quite similar to [3], incorporating CPUs, FPGAs, ASICs and distributed memory into different modules. The architectures of Figures 1b, 1c and 1d are very difficult to be prototyped in real hardware, especially because of the increased cost for acquiring such hardware platforms. Hence, using VP in OpenCL-based systems (i) facilitates cost-free vendor-independent rapid prototyping, (ii) allows for easier and faster platform debugging and timing/power metrics evaluation, (iii) provides extensive architectural flexibility and (iv) enables the iterative platform refinement with a small set of architectural details in each design stage.

VP modeling without OpenCL: A typical VP for heterogeneous MPSoCs requires (a) the software programming, (b) the modeling of computation for hardware accelerators and (c) the modeling of the interconnection. Modifying one of these elements might result in modifications to the other parts as well. For example, re-assigning a task to another processor type might lead to software changes for handling the newly-assigned component. In another example, a bus might need different accelerator modeling than a NoC: in the latter case, the accelerator might be adapted in order to exploit transactions parallelization. Therefore various portability issues arise between different architectural schemes incorporating different processors, memory organization and interconnection schemes. This is alleviated by using OpenCL during prototyping, as OpenCL is able to provide a portable programming and simulation environment which is adapted to any architectural scheme, without the need for software or hardware model modifications, while also enabling the easy runtime assignment of the application tasks onto the processing elements.

III. RELATED WORK

Due to its provided functional portability, OpenCL has been extensively supported in GPU programming, in order to abstract away the complex programming model of GPUs. Typical examples are the OpenCL development environments of AMD [4] and NVIDIA [5]. A survey on the performance and portability of OpenCL in GPUs is provided by [6]. Apart from GPUs, an ever-increasing effort is made for adopting OpenCL in FPGA design. The most typical example is Altera SDK for OpenCL [7], an OpenCL-based development and execution environment that allows for the automatic synthesis of OpenCL code down to FPGA bitstream, while including the appropriate communication environment between the host program and the FPGA-mapped accelerator(s). Xilinx also adopted OpenCL by providing Xilinx SDAccel environment [8], which provides an integrated development and runtime solution from C, C++ or OpenCL sources down to FPGA-mapped applications [9], as well as by enhancing Vivado HLS tool with OpenCL support (however only for high-level synthesis) [10].

Although the above OpenCL-based development environments for FPGA programming are evolving more and more, they suffer from the inherent constraints of FPGA-based system design: The system design is made in RTL, by executing the whole flow which is required in order to (a) transform an OpenCL code into a hardware description and (b) map the description onto an FPGA, e.g. by using Quartus in case of

the Altera SDK for OpenCL. Moreover, the system evaluation is only made by using a real (and potentially expensive) FPGA board. Apart from the cost, there is no explicit support of alternative architectures involving CPUs other than those provided by the SoC fabric of the FPGA board. Last but not least, Altera SDK for OpenCL requires a license for compiling an OpenCL description of the accelerator, while SDAccel can be obtained only after contacting Xilinx.

This paper supports that the above issues can be alleviated by combining the portability of OpenCL with the abstracted hardware modeling of Virtual Prototyping. The most relevant example of such a combination is the OpenCL emulator of Altera SDK for OpenCL, which however is suitable only for functional verification. Moreover, prototyping frameworks that enable the automatic VP creation, e.g. Mentor’s Virtual Prototyping [11], do not explicitly support OpenCL applications, while they also focus on software development. To the best of our knowledge, there is no OpenCL-based framework for virtual prototyping of heterogeneous SoCs in multiple abstraction levels. On the contrary, our proposed prototyping framework addresses the above issues by (a) providing an automated flow for deriving a SystemC-based VP from an OpenCL source, and (b) enabling the VP simulation with different configurations, without the need for existing hardware. The vendor-independent nature of the framework enables the use of numerous different architectural schemes which might be difficult to map onto an FPGA.

IV. SYSTEMC PROTOTYPING METHODOLOGY FOR OPENCL APPLICATIONS

After a brief background of the OpenCL execution model, this section analyzes the proposed prototyping framework. This analysis includes the framework structure and functionality, as well as a prototyping flow for converting a set of OpenCL kernels into VP modules.

A. Background on OpenCL execution model

The proposed prototyping framework is based on the OpenCL 1.0 specifications [12], according to which an OpenCL application consists of two main parts: (a) the host and (b) a number of OpenCL kernels, as the execution model example of Figure 2 depicts. The kernels part is organized as an OpenCL context, i.e. a unified environment which contains the kernels executable (a.k.a. program), the kernel instances (a.k.a. work-items), the utilized devices and the memories. Therefore, the host controls the kernel instances and the respective devices which are included in this context. Each work-item matches to a specific part of the kernels execution, such as a single iteration of a ‘for’ loop or a branch of an ‘if-else’ block.

To define which parts should be executed in each work-item, built-in OpenCL functions return the global and the local index of the work-item. Although the example of Figure 2

1 It is possible to use multiple devices in one context as well.
utilizes one-dimension indices, the developer can use up to three dimensions. The global index distinguishes each work-item from the others. However, the work-items might be organized in work-groups. In that case, the local index is used for identifying a work-item inside a specific work-group. In the example of Figure 2, given \( W \) work-groups, the global index range is \( 0, ..., W \times i - 1 \), while the local index range for each work-group is \( 0, ..., i - 1 \).

This grouping is related to the OpenCL memory model: There are four distinct memory types: (a) the global memory, which is visible by any work-item, as well as the host; (b) the constant memory, i.e. a read-only global memory; (c) the local memory, which is visible only by the work-items of a single work-group (each work-group has its own local memory); and (d) the private memory, which is used only inside a specific work-item. This memory model allows for multiple memory accesses when using local memories for temporary data sharing, thus leveraging the parallelization potential provided by OpenCL. To avoid race conditions, synchronization mechanisms known as barriers can be used inside the kernel code, for global, local or both memories.

Therefore, within a specific context, (1) the host selects the execution of one of the OpenCL kernels and defines a set of buffers for data sending/receiving. (2) After enqueuing\(^2\) the input data to be sent to the global and/or constant memory, (3) the host invokes (i.e. triggers) the kernels by enqueuing an “NDRange” command, which involves the creation of an N-dimensional range of work-items and work-groups. Afterwards, (4) the data and the command are flushed to the deployed devices. (5) When the OpenCL kernel is executed, an event is returned to the host. (6) The host enqueues a command for data reception. This typical flow is repeated for each kernel.

**B. Structure for the Proposed Prototyping Framework**

Based on the OpenCL execution model of Section IV-A, Figure 3 depicts the main structure of the proposed OpenCL-based virtual prototyping framework, which comprises the host and the SystemC-based VP part. The host is either a x86 PC or an instruction-set simulator, e.g. provided by OVP [13]. In both cases, the host software utilizes the OpenCL host API, which provides standardized functions for command/data enqueuing and synchronization. The API manipulates an Inter-Process Communication (IPC) mechanism for the connection with the VP. If the host is a software simulator, the IPC manipulation is made via a Transaction-Level Modeled (TLM) adapter (IPC2TLM), with which the accesses to specific bus addresses are translated to IPC commands. This scheme enables the decoupling of the software simulator from the OpenCL work-items, following the concept of Figure 4: In software simulators like OVP, each platform component is scheduled in serial for a specific time quantum. Therefore, in a single platform including CPU models and OpenCL work-items, the following behaviour is noticed: If a signal is sent from the CPU to the work-items, it will take effect only at the end of the CPU time quantum. The time frame between the signal sending and the end of the quantum will be wasted. This also occurs when the work-items send a signal to another component before the end of their quantum. To provide a simulator-independent solution for this issue, this work proposes the use of a separate software simulator (e.g. OVP-based platform with CPU models and memories, all connected typically via a TLM bus), which runs in parallel with the VP. With this scheme, the signal exchange will instantly take effect, based on the event-driven scheduling of SystemC. In addition, this decoupling enables the VP parameterization during the application execution.

For data exchanging between the host and the VP, a shared memory segment is allocated into the host. This segment includes the global and constant memory for the VP. In addition, the shared memory incorporates a 64-bit variable for the simulated time of the VP. This variable is necessary because, during the application execution, the VP may be restarted in order to execute another kernel or different work-items. Hence, in order to avoid the resetting of the simulated time, the VP time-stamp is stored into the external time variable. This variable is also utilized for time profiling through the built-in OpenCL functions.

The VP consists of multiple work-items\(^3\), which are organized in work-groups. Each work-item is a SystemC-modeled accelerator which includes the OpenCL kernel code, as well as control and data signals. An important feature is the gated clock input for each work-item: Firstly, it enables a low-power system design in early design stages. Secondly, this technique may lead to significant simulation time improvements, as SystemC is enabled to omit the unused (e.g. early-finished) work-items. All work-items are controlled by a wrapper module, written in SystemC, which provides (1) the work-item interconnection, including the data access arbitration and the work-item synchronization (i.e. barrier handling), and (2) the control handling from the host via the IPC, i.e. work-item triggering and event notification. Also, the wrapper includes a pointer to the shared memory segment for global data and constants, as well as local memories, one for each work-group. This VP organization features modularity and configurability: the designer may use different system architectures by only choosing another wrapper version with different interconnection scheme (e.g. bus, Network-on-Chip, etc.) and memory model (e.g. distributed memory, etc.), without having to change the behavioural description of the work-items, and vice-versa.

Below, we provide an analysis on the layout of a typical wrapper and the proposed IPC mechanism.

1) **Wrapper layout**: Although the layout of a work-item wrapper strongly depends on the deployed interconnection and memory model, this section provides a typical wrapper architecture, which can be used as a paradigm for designing a wrapper library with a variety of different architectural features. The wrapper consists of two main modules, which control the VP work-items: (a) the scheduler and (b) the memory and interconnection model.

\(^2\)This term describes the buffering of data and/or commands. The buffered content may not be sent immediately to the device(s), but only when the host reaches a specific synchronization point.

\(^3\)In VP side, the work-items match to the available resources of the platform.
Fig. 5. Work-item wrapper scheduler.

Fig. 6. Work-item wrapper model for interconnection and memory.

i. Scheduler: The host may invoke more work-items than the available resources of the VP. In this case, the scheduler is responsible for the serialization of the invoked work-items, according to the available ones in the VP, as shown in Figure 5. The invoked work-items are separated into parallel groups, the number of which is equal to the number of VP work-groups (i.e. \( W \)). In each group, the invoked work-items are organized into \( S \) segments, in each of which the invoked work-items should not exceed the available ones. The scheduler properly adjusts the global and local indices, so that one segment is running on the available work-items. When the execution is finished, the work-items are re-triggered for the next segment.

ii. Memory and Interconnection: As Figure 6 shows, the wrapper uses separate local and global interconnection for local (one for each work-group) and global data access respectively, thus enabling data access parallelization. Each work-item has dedicated input/output signals for local and global interconnection. Every interconnection is a typical crossbar which consists of input/output ports, one pair for each work-item, as well as one pair for the memories. Each pair of ports consists of control and data signals, allowing for transactions in words of multiple bytes, defined by the designer at compile time. The latter enables single-cycle transfers of vectors of 2, 4, 8 or 16 values (of up to 32-bit each), which are supported by OpenCL [12], thus enabling parallelism on data processing.

Each module of the local memory is attached to one local interconnection, while a global/constant memory is attached to the global interconnection. Upon memory access, the work-item source code defines the memory type (global/constant or local) and the address inside the memory. If multiple work-items access the same memory module, a round-robin arbitration is applied. We assume that single-port memories are utilized, supporting 32-bit accesses. However, significant bottlenecks may be induced, especially when reading global or constant vectors of data. Hence, a cache module is used for the global data\(^4\), the size of which is determined at compile time. The cache supports accesses in lengths equal to the word length of the interconnection, in order to retain the interconnection performance and thus avoid bottlenecks. The area/power cost of such a cache depends on its size and word length, however the designer may fine-tune both parameters for achieving optimized solutions.

2) Inter-Process Communication: The IPC mechanism of the proposed framework is based on a set of Unix semaphores, which are utilized for the control between the host and the VP, as shown in Figure 7. In particular, the set includes three semaphores; one for the VP triggering (i.e. “Trigger”) and two for the host notification when processing ends (i.e. “Ready” and “Ack”). Apart from the semaphore-based control, the IPC mechanism incorporates an API for data exchange. In particular, the shared memory segment is manipulated by the host through `memcpy()` calls. Also, the time variable is updated by the wrapper in every (simulated) clock tick. The host reads this variable when polling the current time-stamp.

Hence, (1) when the host invokes a kernel, the VP process is started by taking as input the number of work-items and work-groups, as well as the input/output data size. During the VP startup, the semaphores and the shared memory are attached to the process. Afterwards, (2) the host triggers the data processing through the “Trigger” semaphore, which is polled by the wrapper. This kind of waiting is non-blocking\(^5\) in order not to stall the simulated time. (3) During processing, the host waits until the result is ready, using the “Ready” semaphore (typically this is a blocking waiting). (4) When the wrapper notifies the host that the processing has finished (through the “Ready semaphore”), (5) the VP process performs a blocking waiting through the “Ack” semaphore, which is used for verifying that the host has received the notification.

C. Prototyping Flow for OpenCL Applications

In order to automatically create the work-items prototype, the proposed framework is accompanied with an OpenCL-to-SystemC prototyping flow, presented in Figure 8. After a syntax check (typically using `clang` [14]), the OpenCL source is converted into SystemC by using (a) a work-item template; (b) a C++ class for OpenCL vectors\(^6\) of different data types, supporting arithmetic/logic operations and vector comparisons according to the OpenCL specifications [12], while also enabling different degrees of parallelization in vector processing; (c) mathematical functions for both scalar variables and vectors; and (d) input/output functions.

As the syntax of OpenCL vector operations differs from the default C/C++ syntax, any vector-related operation is rewritten according to the provided methods of the deployed C++ vector class. Figure 8 (“Vector Processing”) shows typical conversion examples. This conversion is applied recursively: For example, `Vload` is firstly converted to `V(s)`; then to `Vector(V(1),V(3))` and finally to `Vector(Varray[1],Vararray[3])`.

\(^4\)The constants are fetched only once and are saved inside the work-item.

\(^5\)In non-blocking waiting, the process is not blocked, but it performs active waiting. In blocking waiting, the process is blocked.

\(^6\)Different from the built-in vector class provided by C++. 

---

1. Scheduler
2. Memory and Interconnection
3. Inter-Process Communication
4. Cache module
5. Non-blocking waiting
6. Different from the built-in vector class provided by C++.
Fig. 8. Proposed prototyping flow.

Additionally, the OpenCL-to-SystemC conversion includes the detection of the global constants and the global and local variables\(^7\). Every access to that data is replaced by input/output function calls for implementing memory accesses to/from the memories, as shown in Figure 8 (“I/O Transactions”).

When the SystemC source for the work-items is created, the next stage of the proposed flow involves the construction of the whole VP, including the wrapper model. Finally a conventional C++ compiler is utilized, in combination with the SystemC library, so that the VP executable is produced.

The main advantage of this flow is that it is applied only once: The OpenCL code is not needed any more in system design, as the created VP will be utilized in all the remaining hardware design stages, i.e. (i) functional verification, (ii) design parameters evaluation in terms of timing, resource utilization and power consumption, as well as (iii) final synthesis. This is also a major contribution of this work. To better explain this advantage, Figures 9a and 9b show two typical design flows when using (a) the proposed prototyping framework and (b) Altera SDK for OpenCL respectively. Altera SDK for OpenCL is chosen as state-of-the-art for mapping OpenCL kernels onto FPGAs.

The design flow using the proposed framework (Figure 9a) starts with the prototyping procedure of Figure 8. The produced VP can be used in a typical design space exploration (DSE) and can be refined with more architectural details in later design stages. In a single DSE, the VP is enriched with timing/area/power annotations which are derived using High-Level Synthesis (HLS) [15]. The annotated VP is then compiled and simulated. During simulation, the computation and communication behavior are combined for timing and power estimation, while different execution scenarios are taken by using different input data.

As compared to the above flow, the design procedure using Altera SDK for OpenCL (Figure 9b) starts with the OpenCL kernels compilation, including HLS and RTL synthesis with Quartus. The result is a bitstream for programming an FPGA board, where the system is evaluated. When using a typical DSE, the whole procedure is repeated after every parameters change. Also there is no support for higher abstraction levels.

V. EXPERIMENTAL RESULTS

The proposed prototyping framework is evaluated by using Rodinia benchmark suite [16]. Rodinia provides OpenCL-implemented algorithms, mainly focusing on GPU acceleration. However, the provided OpenCL kernels can be mapped onto FPGAs as well. For the scope of this work, we used the benchmarks of Table I, which also shows the number of work-items in total (including the invoked and the available work-items in the VP) and locally (i.e. per work-group), as well as the application input size. In applications with large number of invoked work-items, the OpenCL kernels have been partially serialized, in order to both avoid the excessive memory allocation and provide a more realistic model of the system-under-design.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Invoked Work-Items</th>
<th>Available Work-Items</th>
<th>Local Work-Items</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastLambert</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>2 × 128 matrix</td>
</tr>
<tr>
<td>BFS</td>
<td>1024</td>
<td>1024</td>
<td>256</td>
<td>1024-node graph</td>
</tr>
<tr>
<td>Gaussian Elimination</td>
<td>64 × 64</td>
<td>64 × 64</td>
<td>16 × 16</td>
<td>matrix</td>
</tr>
<tr>
<td>Particle Filter</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>256 video particles(^a)</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>42816</td>
<td>64</td>
<td>42764</td>
<td>records</td>
</tr>
<tr>
<td>Histogram1024</td>
<td>384</td>
<td>10000</td>
<td>98</td>
<td>10000 elements</td>
</tr>
<tr>
<td>MergeSort</td>
<td>3072 × 1024</td>
<td>256 × 1</td>
<td>10000</td>
<td>elements</td>
</tr>
<tr>
<td>BucketSort</td>
<td>1024</td>
<td>128</td>
<td>10000</td>
<td>elements</td>
</tr>
<tr>
<td>Back-Propagation</td>
<td>16 × 64</td>
<td>16 × 64</td>
<td>64-input ANN(^b)</td>
<td></td>
</tr>
</tbody>
</table>

\(^a\) In 10-frame 32 × 32 video
\(^b\) Neural network with 64 inputs, 1 hidden layer with 16 neurons and 1 output

\(^7\)Private variables are finally implemented as registers.

\(^8\) Altera does not provide a standalone HLS tool. However, despite the use of different vendors, we intend to acquire typical execution time results only. Similar HLS run-time results are expected with the use of any other commercial HLS tool.

\(^9\) The board execution time is multiplied by 500× to be visible to the chart.
As Figure 11 depicts, the use of an OVP software simulator as a separate platform is able to leverage the high simulation speed provided by OVP without causing significant communication overhead between the CPU and the OpenCL work-items, as it achieves similar simulation times as compared to the use of a x86 host. On the contrary, the use of a single platform may cause significant simulation time overhead, ranging from 10% up to 5×. The first reason, as explained in Figure 4, is that a single OVP-based platform performs quantum-based OVP scheduling, which may lead to significant time waste. The second reason is that a single platform deploys a constant number of work-items. On the contrary, in separate platforms, only the necessary work-items are allocated. Thus if the host repeats a kernel with less invoked work-items, the VP simulation will be faster, as less components will be simulated.

VI. CONCLUSIONS

This paper presents a rapid prototyping framework, which automatically derives a SystemC-based VP from OpenCL sources, thus combining the OpenCL portability with the abstracted modeling of Virtual Prototyping. The proposed framework supports different hardware architectures and memory models without the need for kernel modifications, while also enabling fast evaluation cycles, without long compilation procedures. In particular, the design flow which accompanies the proposed framework achieves evaluation time improvements up to 18×, as compared to Altera SDK for OpenCL. The proposed framework also enables the use of any OpenCL host, which can be either a x86 CPU or a software simulator. The host communicates with the VP through an inter-process communication mechanism, which also allows for the separation of a software simulator from the VP, thus leading to significant simulation time improvements reaching up to 5×.

REFERENCES