2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Open Source Virtual Platforms for SW Prototyping on FPGA based HW

Chen Qian, Nvidia, Shanghai, China (cqian@nvidia.com)
Praveen Wadikar, Nvidia, Bangalore, India (pwadikar@nvidia.com)
Mark Burton, GreenSocs, Chalagnac, France (mark@greensocs.com)

***Abstract*— Meeting the needs of both simulation speed, and hardware representatively is always challenging. A novel combination of virtual platform technology is introduced that takes advantage of existing infrastructure (including GreenSocs QBox, and existing FPGA availability) allowing high performance, but accurate simulation.**

Keywords—Qemu, SystemC, TLM, FPGA

I. Problem Statement:

Deep Learning(DL) has hit the headlines around the world; NVIDIA leads the industry in DL, in which, training and inference are two critical aspects [1]. To maintain leadership, NVIDIA's Deep Learning Accelerator (NVDLA) is provided in open source as an open architecture that promotes a standard way to design deep learning inference accelerators [2].
Both software(SW) and hardware(HW) are open source, but there is a major restrictions: There is no way to run software directly on potential hardware designs. Software development must wait for hardware to be designed, then run it on an FPGA or wait for real silicon. Hence hardware changes take a huge amount of time to propagate to software, this turnaround, from hardware changes to software might take weeks or months.
This will inevitably reduce the competitiveness an offering such as the NVDLA. To overcome these issues, an open source virtual platform (OpenVP) has built along with the NVDLA (which is, itself available openly [3]).

The critical features of this virtual platform are the ability to quickly prototype hardware designs, and for the execution speed to be sufficiently fast to enable software development. Because of the nature of the Deep Learning Accelerator – being an 'accelerator' – the speed of the simulation is absolutely critical. Addressing this concern is the major focus of this paper.

II. Importance:

The open source NVDLA provides a standardized (based on SystemC and TLM-2.0, part of the IEEE 1666 standard [4]), open architecture to address the computational demands of inference. Any new DL idea invariably involves both complex HW and SW transformations; the advantage of an open source project is that users from all over the world can prototype their ideas. This is critical for complex DL projects. Having a 'standard' platform that provides capabilities to prototype both HW and SW ideas quickly, allowing HW updates to be reflected in SW quickly, is becoming a requirement.

The DLA accelerates a number of algorithms common to Deep Learning, for a simulation to be of value, it must also exhibit this ability to accelerate these common algorithms.
Typical 'virtual platforms' rely heavily on the host machine to execute the functionality of a device that is being virtualized (or modeled) [5]. In the best case some of the functionality can be 'offloaded' onto the host machine (hence a model of a graphics processor may use a graphics library which in turn makes use of the host machines graphics processor to perform the algorithms [6]). In the general sense this is making use of Hardware in the simulation - so called Hardware In the Loop (HIL) [7].

In the case of new Deep Learning algorithms, that 'host functionality' is typically not present. Relying on the host CPU itself to perform the algorithms shows no advantage to the software engineer. Further more – implementing the proposed Deep Learning algorithm not only in e.g. RTL, but also as a highly optimized model can take time, which further delays the development cycle.
One answer to this quandary might be to use FPGA's. This is the approach taken by this paper.

III. PRIOR WORK:

The ‘standard’ that addresses the area of Virtual Platforms is SystemC (IEEE 1666) [4]. This standard provides the language in which virtual platform components will be written (for instance by users of the OpenVP). An open source version of SystemC is available as Accellera’s proof of concept simulator. SystemC does not provide core CPU models, it simply defines the language that must be used.
There are many open source simulators that provide a wide range of CPU models. One of the most developed and supported is QEMU [8,9]. In common with other such simulators, it must be integrated into SystemC. Greensocs provides such an integration with a QEMU variant named QBOX [10]. This also supports multiple CPU instances, and the use of multiple host threads. For the complex devices that are used in the NVDLA, this is a critical feature to achieve good simulation speed.
In addition to the simulator itself, in order to provide the representatively of real HW and the speed required, the solution presented here also makes use of “Hardware In the Loop” (HIL). In this case, directly using FPGA’s. This allows the real RTL design to be part of the simulated OpenVP.

Co-simulation, or co-verification [11] is a technique that has existed for decades, allowing a model of one component to execute in one environment, while another model executes in another environment. This has been applied to CPU models executing at the level of the functional instruction set, connecting to RTL models executing at the clock level. More recently CPU models have been connected to models of the RTL, executing at higher levels of abstraction [10]. Accellera defined two levels of abstraction (Loosely timed and Approximately timed) [4]. For software development (the main aim of this work), the LT level is more appropriate. Nonetheless, this has required Co-simulation techniques to be deployed to allow the integration of the models. Those techniques are equally applicable to models at the LT level, or indeed at the RTL level, which raises the possibility of including HIL.

In the past HIL integration has been fraught with difficulties because the hardware itself typically has a number of timing requirements that the model struggles to fulfill [12]. Our solution to this problem is to treat the hardware as an accelerator to the model, rather than a component in its own right. In doing so, we are able to alleviate the hard real time problems that have beset previous solutions.

Hence we also build on developments in the integration of FPGAs into general computation [13,14]. This is driving performance improvements for cognitive computing and AI, general high performance computation and IoT applications [15]. Today companies such as Amazon and Microsoft provide simple to use services with a defined API to FPGA services [16,17].

IV. TECHNICAL SOLUTION:

Ideally a solution is required that allows the same SW to run on several different environments, from the virtual platform through to the final physical hardware.

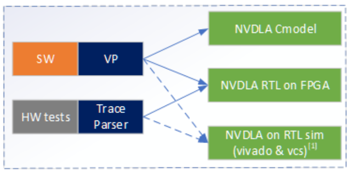

Figure 1

The OpenVP allows SW drivers to run on NVDLA’s C-model and the FPGA as shown in Figure 2. In our case, our application level runs under a Linux guest, and uses a Kernel Mode Driver (which is the critical software driver). The KMD, the application and the User Mode Driver are totally unchanged and the same binary is expected to run on the physical hardware.

The hardware itself is modeled using SystemC and the GreenSocs QBox (based itself on QEMU), which already supports booting Ubuntu with ARMv8 cores.

QBox is capable of containing both CPUs and other devices within the QEMU framework. In our case, a complete ‘virt’ machine is emulated inside QBox. The ‘virt’ machine does not correspond to any specific hardware, but it a useful ‘virtual machine’ which contains all the required services for the operating system. To this kernel, we add the KMD to support our application and UMD level.

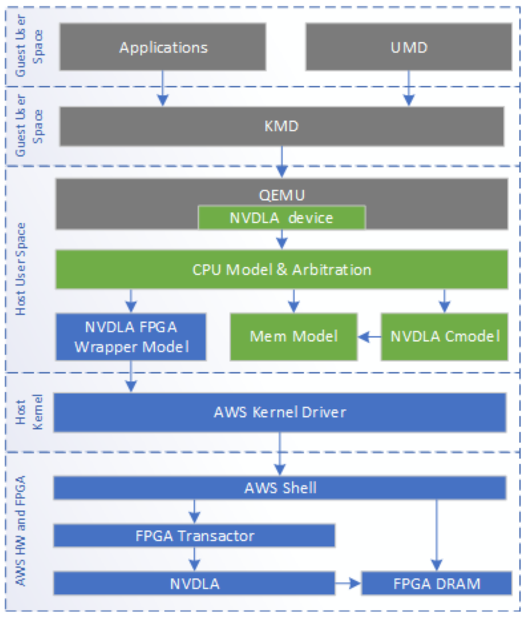

Figure 2

For hardware that is not modele routed to SystemC. The QBox behaves like a standard TLM-2.0 LT model, specifically making use of the ‘quantum’ mechanism defined in TLM-2.0 to handle synchronization. QBox provides a number of different ‘syncronisation algorithms’ based on the Quantum, including allowing Qemu and SystemC to run as close to real-time as possible, or in lock-step, or deterministically, or within a ‘quantum’ of each-other. This latter approach is the ‘typical’ approach in TLM-2.0 models, however the GreenSocs approach allows both the QEMU and SystemC frameworks to run in parallel on separate host threads. This is important in two respects, first it permits good performance (even when using just a C-model of the NVDLA), and second it allows the integration with the FPGA framework to work in parallel while the CPU model is busy.

The methodology, which uses a system of asynchronous events is introduced to overcome the communication issues in hybrid simulation. This methodology allows communication and synchronization between both QEMU and SystemC, but it could work between SystemC and any external simulation framework (even other SystemC’s). However, it relies on the ability to ‘suspend’ SystemC when (and if) SystemC’s time runs too fast and drifts ahead of the external simulator(s). The GreenSocs approach manages this using a single central semaphore, which, to allow composability, needs to be part of the SystemC standard. GreenSocs is currently working with the support of a number of organizations to try and standardize this central mechanism. Interestingly, the same mechanism is also required for save-and-restore, which is also a topic of great interest to the Accellera SystemC LWG.

Within SystemC, the NVDLA C-model is integrated as both an initiator and target such that it may both receive and initiate memory transactions, as it functions directly on ‘bulk’ memory (shared with the main processor). Hence with this C-model of the NVDLA, SW drivers can run real DL networks. However the model has significant performance issues. This is a fundamental limitation of using the host to ‘emulate’ the functionality of the accelerator as discussed above. While the C-model itself is a relatively efficient implementation of that functionality, it will never approach the performance of dedicated hardware.

The C-model is just a high-level implementation of the NVDLA, which enables SW prototyping. By replacing the functionality and connecting to an FPGA, the performance issues can be addressed, and algorithms can be tested on the real design.

For this, as it provides easy access, the Amazon Web Service (AWS) FPGA infrastructure has been integrated into the OpenVP. When the FPGA mode is enabled, an FPGA wrapper model is instantiated to replace the NVDLA and Memory C-model. From the outside, this ‘wrapper’ looks exactly like a standard TLM model. Within the wrapper model, a software transactor driver is created to drive the AWS FPGA with both the NVDLA and other FPGA components installed. As the FPGA will be working on memory based data, the data memory is also held within the FPGA. This is presented in the memory map and appears as a standard TLM model. Essentially, depending on whether the FPGA is being used or not, a ‘mutex’ sends memory accesses one way or the other.

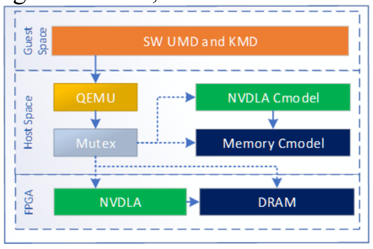

Figure 3

There is a cost to send and receive information via the AWS FPGA interface; While this is very large, the benefits of the FPGA outweigh the costs (by a considerable margin).

The complete solution allows UMD and KMD software to be executed on real RTL, on an FPGA, or on a C-model. H/W trace vectors can also be injected to further test the design.

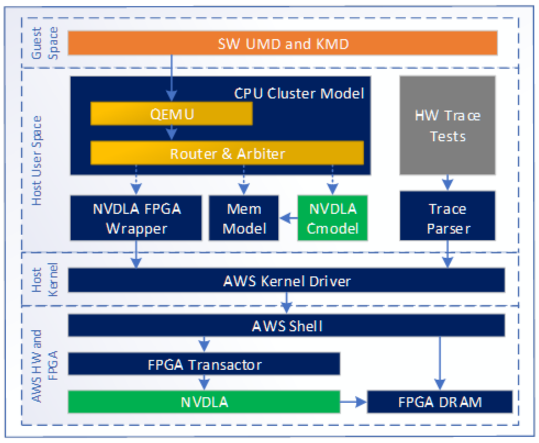

Figure 4

V. QUANTITATIVE BENEFITS:

The initial version of OpenVP, including GreenSocs QBox has been released along with the first NVDLA open source version, and the OpenVP FPGA mode is also available from github.

Using FPGA’s in this novel configuration promises two separate, but complimentary advantages: first, to improve the time to have a model in place, second, to improve simulation performance itself.

As mentioned in the introduction, the time taken to update an algorithm in the DLA, have that reflected in an environment that the S/W team can use, and finally have the S/W adjusted is critical. Previously this has taken weeks or months.

As the OpenVP is available, NVIDIA’s NVDLA SW team has been using it as a sign-off environment for their open source software. The SW team can start testing the latest configuration and network design (e.g. Alexnet) directly on the OpenVP. In one day the HW itself, or the C-model can be updated and SW re-configured. This compares to e.g. a one-week latency previously. When algorithms are directly synthesized into the RTL (which is increasingly common), providing a direct route from there into simulation reduces algorithmic-change to simulation availability to minutes.

Secondly, of course, the speed of simulation is critical. Because the SW uses the latest HW design through the FPGA, the simulation speed of the full network goes from hours in the C-model to minutes on FPGA.

Performance	Cmodel	FPGA
nv_small	6,128s	884s (4s for DLA)
nv_large	4,000s	661s (1s for DLA)

Figure 5

Figure 2 shows two example models (nv_small and nv_large). Both show over a 6 fold increase in performance by using an FPGA. However, note that while the actual FPGA algorithm code was busy for 4 and 1s respectively the overall simulation took 884 and 661 seconds. The overhead represents the time that the rest of the platform consumed, and the communication between the rest of the platform and the FPGA. We believe that this could be further optimized.

VI. CONCLUSION

While complexity increases, accelerating software using domain specific devices is becoming ever more popular. However the turn-around between designing a new acceleration algorithm to finally having completed software that takes advantage of that algorithm can be large. This paper presents a technology that combines existing developments in Virtual Platforms (From GreenSocs), combined with the now-available FPGA cloud offerings (AWS) to provide a complete solution (for NVIDIA's NVDLA platform) which enables much faster simulation speeds, but also an automatic path from algorithm to simulation, allowing software to be developed much earlier. NVIDIA's software team now use this platform as a matter of course, reducing turn-around times to hours rather than weeks.

VII. REFERENCES:

1. *"Nvidia CEO bets big on deep learning and VR". Venture Beat. April 5, 2016.*
2. https://github.com/nvdla
3. https://github.com/nvdla/vp
4. https://standards.ieee.org/standard/1666-2011.html
5. *Graziano, Charles. "A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project"*
6. INTEGRATE AND VERIFY SYSTEMC MODELS IN A GRAPHICAL ESL TESTBENCH, Jerome Lemaitre, CoFluent Designs and Dr. Mark Burton, GreenSocs http://chipdesignmag.com/display.php?articleId=2440
7. https://dvcon-europe.org/sites/dvcon-europe.org/files/archive/2014/proceedings/T4_3_presentation.pdf
8. QEMU, a Fast and Portable Dynamic Translator, Fabrice Bellard, http://archives.cse.iitd.ernet.in/~sbansal/csl862-virt/2010/readings/bellard.pdf
9. https://www.qemu.org
10. QBox: an industrial solution for virtual platform simulation using QEMU and SystemC TLM-2.0 Guillaume Delbergue, Mark Burton, Frederic Konrad, Bertrand Le Gal, Christophe Jego. ERTS 2016
11. A Framework for Fast Hardware-Software Co-simulation ,Andreas Hoffmann, Tim Kogel, Heinrich Meyr. DATE 1991
12. Real-Time and Hardware-In-The-Loop Simulation of Electric Drives and Power Electronics: Process, problems and solutions Simon Abourida*, Christian Dufour*, Jean Bélanger, International Power Electronics Conference 2005
13. https://thenewstack.io/developers-fpgas-cloud/

14. J. Weerasinghe, R. Polig, F. Abel, “Network-attached FPGAs for data center applications,” in IEEE International Conference on Field-Programmable Technology (FPT ’16), Xian, China, 2016.
15. https://www.eetimes.com/author.asp?section_id=36&doc_id=1330431#
16. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/fpga-getting-started.html
17. https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-accelerate-with-fpgas