Chapter 8
Enabling Virtualized Programmable Logic Resources at the Edge and the Cloud

Kimon Karras, Orthodoxos Kipouridis, Nick Zotos, Evangelos Markakis and George Bogdos

8.1 Enhancing Compute Through Programmable Logic

Continuous, unabated performance increases are an innate part of the computer world from the invention of the first microprocessor until today. This trend appears to be, however, slowly but steadily coming to an end. Single thread performance gains have been slow to extract from each new technology node for more than 10 years. This was countered successfully by a massive push into parallel computing in the form of multi-core, multi-threaded processors which continues until the present, albeit while showing clear signs of running out of steam. The reason is that parallelism isn't a silver bullet that can be applied to every problem in infinite amount. Many common applications can't be parallelized further, and thus, any gains that can be extracted by increasing the number of cores and/or threads are limited to only small subset of all applications.

This has led academia and industry alike in search of a solution that will allow for the processing of the massive heaps of data that are being collected throughout the world by sensors, cameras, the Internet, etc. The most promising fruit of that research has by far been heterogeneous compute, a concept that encompasses anything from

K. Karras (✉) · O. Kipouridis · N. Zotos · G. Bogdos
Future Intelligence Ltd., London, UK
e-mail: kkarras@f-in.co.uk

O. Kipouridis
e-mail: akip@f-in.co.uk

N. Zotos
e-mail: nzotos@f-in.co.uk

G. Bogdos
e-mail: gbodgos@f-in.co.uk

E. Markakis
Technological Education Institute of Crete, Heraklion, Greece
e-mail: markakis@pasiphae.eu

© Springer International Publishing AG, part of Springer Nature 2019
C. Kachris et al. (eds.), *Hardware Accelerators in Data Centers*,
https://doi.org/10.1007/978-3-319-92792-3_8

CPUs with different core types on the same processor die to modern SoCs that include CPUs, GPUs and custom accelerators or custom processors like Google's Tensor Processing Unit [8] or Micron's Automata Processor [13].

These solutions are trying to address the processing of massive data, principally in the cloud but also increasingly at the edge, where a big part of that data is being generated. Both of these fields are ripe for exploration where a multitude of concepts are vying for domination. Regardless of the details of each architecture, the basic premise and ingredients are always the same, as are the issues plaguing the current cloud-centric paradigm which they're trying to solve, namely:

- Heaps of data being directed to the cloud for processing and/or storage, leading to congestion at the network leading to and from the data centers.
- Unacceptably high decision-making latency leading to delays in taking action based on collected data.

The former is a direct result of the centralized architecture, which treats edge nodes as simple data collection points and/or actuators and ships all the data to one or more massive data centers for analysis. As the number of edge devices explodes and the data they collect diversifies, the traffic being fed into data centers will be strained to the extreme, which means that either the network will become congested or unacceptably high investment into infrastructure will be required to avoid this. Congestion on the path to and from the data center will unavoidably lead to the second issue, latency. Decision-making latency means the time it takes to reach a decision regarding an action to be taken and implement that action wherever required, usually at the networks edge. Currently in order for this to happen, data needs to be sent up to the cloud, sifted through and the decision needs to propagated downward to the edge again over multiple, potentially unreliable hops. Thus, it is clear that there are tangible benefits to accelerating tasks into the cloud but also to shifting parts of the processing load toward the networks edge. This translates into increased processing requirements for small, low power mist and fog nodes, which are currently not up to the task and which can't simply use the powerful CPUs found in contemporary data centers due to power constraints. This makes the case for compute acceleration that delivers significant performance gains at low power consumption at the edge even more prescient than in the cloud.

One alternative that can offer impressive performance increase while maintaining very low power consumption, a feature critical in power-strapped edge nodes, are FPGAs. There's a long literature of examples where FPGAs provide impressive performance benefits in comparison to standard CPUs, which Sirowy and Fiorin sum up excellently [11]. At the same time, FPGAs can be reprogrammed quickly and infinitely so the tasks assigned to them can vary over time making them easily adaptable to changing demands and capable of receiving updated dynamically in the field. That being said, programmable logic has been marred by several caveats that hinder its adoption:

- While FPGAs are reprogrammable, the process does not allow for automated, remote deployment of tasks from a distance. Typically, programming has to be performed with a cable attached to the device.

- FPGA development requires very esoteric knowledge of the target device including low-level details like pin assignments and interface standards.
  This chapter proposed the Programmable Compute Platform (PCP) which addresses and alleviates both of these concerns by offering:
- A system consisting of both HW and SW that allows for the integration of an FPGA into the edge and/or cloud and the remote (re)deployment of tasks to it over a standard SW stack.
- A working hardware system with strictly defined, intuitive interfaces where tasks can work in a plug-and-play manner thus reducing the overhead to develop tasks for remote deployment.

In the current iteration of the platform, this is accomplished by utilizing a novel device called an FPGA SoC. We use this device to achieve an optimal division of labor between the A9 CPUs and the FPGA, with the CPUs executing the SW that enables the remote deployment of an HW task (similar to an HW virtual machine) on the programmable logic. This includes an agent that can reply to queries about the devices state and can receive a HW VM image and deploy it to the HW and/or create virtual network interfaces to integrate the device seamlessly into service function chains.

Our system is at the stage of a functioning proof of concept with all of the mandatory HW and SW components up and running on the Xilinx ZC706 development board. An h264 decoder is used as a test HW VM to highlight the benefits of using an FPGA as an accelerator and the performance and power advantages it brings.

The remainder of this chapter is organized as follows: Section 8.2 provides an overview of existing efforts in this area. Section 8.3 introduces the programmable cloud platform and its components, while Sect. 8.3.3 explains the changes made to the OpenStack controller to make it FPGA-aware. Section 8.4 provides preliminary results both for the platform and the HW VM being tested, and finally, Sect. 8.5 summarizes our findings and provides pointers for future work.

8.2 Prior Efforts in Virtualizing Programmable Logic

Research in integrating FPGAs in the existing cloud infrastructure is scarce though the topic has been gaining traction as of late. Edge acceleration has on the other hand been flying completely under the radar so far, even at a conceptual level. On the cloud front, the handful of research papers can be found on the subject is summarized here, with all of them converging on a single approach:

- Extending OpenStack Nova to allow for the deployment of FPGA-based systems.
- Provision of a static area in the FPGA to allow for the deployment of the NFs.
- A dynamically reconfigurable area in the FPGA in which the user will be able to deploy his NFs.

More specifically, Buma et al. [3] introduce a framework for deploying the so-called Virtualized FPGA Resources (VFRs) to an FPGA which focuses on the HW

side of the problem. They take advantage of the OpenStack service called glance to deploy VM packages which contain FPGA bitstreams to the device. These are then picked up by an agent (running in SW on the compute node), which unpacks the VM and deploys the bitstream to the device. The advantage of this approach is that the FPGA is completely abstracted from OpenStack. Deployment of a VM from its perspective remains does not change in the slightest since the whole process is handled in the agent on the compute node. The agent manages the FPGA, which is divided into a static and a dynamic part. The static part contains all the modules necessary for the deployment of the VFRs and for the communication between the external world and the VFRs (including a network and a DRAM interface). It also performs the necessary handshaking between the VFR and the agent to start and stop it when deploying or retracting it. The dynamic area is where the VFRs are actually programmed. A NetFPGA-10G board is used for a simple prototype, which uses a load balancer application to demonstrate primarily the feasibility of the concept, but also the performance gains that can be reaped by migrating from an SW VM to an FPGA-based one.

Another similar if somewhat more comprehensive effort was published by Chen et al. [4]. This work includes many common elements with the system to be developed within this project. More specifically, it uses adapted OpenStack to deploy HW VM images to the FPGA in which it implements a layered architecture that enables the programmable logic in the FPGA to run these HW VMs in isolation. On the HW level, the programmable logic can be both space- and time-shared by introducing the notion of a segregated accelerator region in which the HW VMs are slotted in. Additionally, a static area is reserved on the programmable logic, which regulates access to DMA resources among the accelerators and manages jobs in them. On top of this HW infrastructure, there are three SW layers that facilitate the deployment and control of the HW VMs from host SW whether that is the cloud controller or SW running locally on the processor. The main limitation of this work is the limited connectivity offered to the accelerators (only a DMA connection to DRAM) and the fact that there is no provision for HW/SW VMs. Furthermore, the paper does not go into detail about how OpenStack had to be extended to support FPGA-based VMs.

A third approach for deploying VNFs to FPGA-based systems was published by Ge et al. [7]. This work shares a lot of commonalities with the [2] in that the programmable logic is divided into slots for accelerators to which tasks are dispatched. The accelerators are slotted in using partial reconfiguration which is handled by a VM running on standard x86 processor. Additionally, each accelerator is deployed over its own SW VM via OpenStack. This means that having an SW component is mandatory even if its just a shell for deploying an accelerator. In the system described it appears that both OpenStack and VMs are not tightly integrated into this system, meaning that the heterogeneity of the system is hidden from these components.

Another indispensable element of any virtualized system is the Hypervisor. Of particular interest for this work are potential Hypervisors that can be executed on the ARM A9 processors found on an FPGA SoC. One possible alternative was presented by Pham et al. [10]. The system introduces in this paper encompasses a wider system

than just the Hypervisor. That system is based on the time-multiplexing of tasks on the programmable logic to accelerate SW executed on the Hypervisor. To accomplish this it creates an overlay onto the programmable fabric to allow for coarse-grained core design and deployment. The most interesting part of this work is the Hypervisor which runs on the ARM A9 cores and is based on the Codezero Hypervisor from Bell Labs. This could be reused in our effort, granted that the source code is available; however, modifications will most certainly have to be made to:

- Allow it to run on the ARM A9s found on the Zynq.
- Allow it to virtualize the FPGA resource.
- Communicate in a compatible manner with OpenStack Nova.

The effort required by each of these changes remains to be evaluated.

It's clear from the short overview provided in this section that while all existing solutions appear to agree on the basic features required for FPGA cloud integration, all of the platforms end up leaving something to be desired. PCP rectifies this by ticking all the boxes and offering a straightforward, intuitive FPGA-based device for the cloud end the edge.

8.3 A Virtualizable, Remotely Manageable FPGA SoC Platform

This section provides a detailed overview of the FPGA SoC-based programmable cloud platform. It consists of a hardware and a software component that together allows for the remote deployment of hardware tasks on the FPGA by using standard cloud management software, namely OpenStack. The hardware component is further elaborated upon in Sect. 8.3.1. The software running on the FPGA SoC's ARM CPUs allows the integration of the platform into OpenStack and is explained in Sect. 8.3.2. Finally, the changes that were performed in the OpenStack controller in order to allow it to utilize programmable logic-based devices are highlighted in Sect. 8.3.3.

PCP makes some concessions in order to simplify the design and strike an optimal balance between design complexity (and thus the overhead of managing the VMs' deployment) and the flexibility offered to the user. The first of the two decisions that were made were to only allow one HW VM to be deployed on the programmable fabric at any given time. This was done to avoid the complexities of parallel deployment of HW tasks on the FPGA. These have been discussed in detail in the past [1, 2, 9] and always led to unmanageable overhead and complicate the design of the hardware tasks for the system excessively thus negating part of the advantages of such a platform. The second decision was to allow for only pure HW VMs to be deployed. Since an FPGA SoC consists of a pair of ARM A9 CPUs and programmable logic, it is in principle possible to have VMs that are made up of a software part running on the ARM and a hardware part running on the programmable logic. The reason this was rejected for the PCP this is that the ARM A9 CPU currently available in FPGA SoC devices is not directly virtualizable itself, which increases the complexity of the task and reduces its appeal since the processors in the PCP already had their work cut

out for them. Future FPGA SoCs like Xilinx's MPSoC [15] will include four ARM A53 CPUs which pack more punch and are directly virtualizable, which will make HW/SW VMs a more interesting endeavor.

8.3.1 Hardware

PCP allows for the deployment of tasks onto the programmable logic, which can be swapped in and out by the OpenStack controller as required. To accomplish this, we utilize a feature present in modern FPGA called dynamic partial reconfiguration (DPR). DPR allows part of the programmable fabric to be reprogrammed on the fly, while the remainder of the device continues to operate as intended. The PCP uses DPR to update the HW VM that is executed on the programmable logic on the fly. As such, the programmable logic in the programmable cloud platform is further subdivided into two parts, the static and the dynamic one, a typical arrangement in systems that utilize dynamic partial reconfiguration to change the functionality of a section of the device. This division is reflected in Fig. 8.1.

The static area is ancillary and enables the reconfiguration of the dynamic one, as well as the interconnection of the processor system (PS) with the dynamic area. It contains an AXI interconnect and AXI DMA module which connect the PS to the programmable logic (PL) as well as the PCAP that performs the reconfiguration of the dynamic area. The AXI DMA module contains two channels: one for data traffic to and from the HW VM and one used for sending monitoring data from the VM to the monitoring SW on the CPU. It doesn't implement Scatter–Gather DMA functionality as this would lead to higher resource usage due to the additional complexity and the

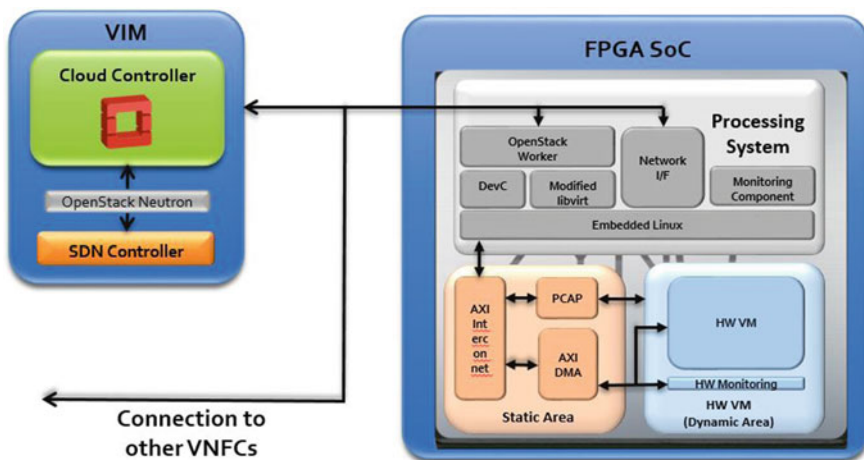

Fig. 8.1 Overview of the programmable cloud platform hardware/software architecture

expected benefit for our system is infinitesimal. The AXI communication fabric runs at 150MHz and has a width of 64 bits thus providing enough bandwidth to and from the dynamic area. An important goal for the static area is that it remains as small as possible to allow for the bulk of the programmable resources to remain available for the HW VMs. In the PCP the static area occupies approx. 6000 LUTs and 3500 FFs, which is less than 4.5% of the available resources on the used FPGA SoC device

The dynamic area is the larger part of the PL and is where the HW VM is deployed after receiving it over the OpenStack worker. The HW VM that is being deployed contains the user accelerator and a monitoring component as mentioned previously. This component is provided as part of the platform; however, the accelerator designer must feed it with the appropriate monitoring data which will then be send to the PS.

8.3.2 Software

A large part of the innovation in the programmable cloud platform comes from its software. At the core of the software is the modified, programmable logic-aware OpenStack (compute) worker node which is assisted by low-level software that interfaces with the Xilinx DMA driver, the DevC driver that feeds data into the PCAP, an FPGA component that is responsible for performing the actual DPR, and a pseudo libvirt library that poses as a counterpart to the worker and answers queries regarding the available resources of the device through an extended API as described in Sect. 8.3.3. As the FPGA SoC in the PCP is single-tenant, the libvirt is a, respectively, simple component. From a high-level perspective, workers executing a modified version of the compute service client are responsible for managing the VMs. Quite similar to its original purpose, workers can deploy, terminate, and reboot VMs, as well as run monitoring services to provide valuable information such as FPGA resource utilization. Given the simplicity of HW VM deployment in PCP, this module is at this stage, respectively, simple but it is bound to grow more complicated as the platform develops.

Additionally, the worker can take of the migration of an HW from one FPGA to another. This is a nontrivial exercise in an FPGA-based VM, as the entire configuration of the programmable logic has to be read out of the device, and then send to a remote node which will redeploy this to a different FPGA. In order for this to happen, execution of the HW VM has to halt first, which means that the worker stops accepting new data for this host, waits for all processing to complete, and then starts reading out the FPGA configuration, which is packaged and then passed on to the OpenStack controller over the network for deployment on a different node. The redeployment process is identical to the deployment of a new VM from a worker perspective. It is worth noting that since programmable logic configuration are completely bound to a specific device type, an HW VM that has been read out from a specific Zynq device (say the Zynq 7045 used in the PCP) can't be automatically deployed to a Zynq 7030 but only to a different Zynq 7045. This limits the flexibility of the migration process, but is a fundamental limitation of how configuration files for FPGAs work.

Furthermore, an overlay to the Xilinx DMA driver has been written which takes care of all the low-level driver functions necessary to create and initialize the DMA channels for reading and writing to the device and presents a character device driver in order to simplify data copying from the OpenStack worker to the PL. Special care has been given to make this a zero-copy driver to maximize the impact of the SW stack on performance.

An important aspect that must be taken care off in the software stack is the synchronization between data traffic and the deployment of a new VM. It is imperative that when the VM is retired and the programmable fabric is to be reconfigured, all pending transactions between the VM and the SW have been completed. This includes all DMA transactions for both the actual accelerator and any monitoring data that might be transferred at any given time. If stale data remains in the AXI interconnect, then this will lead to unpredictable behavior from which it might be impossible to recover. The SW stack takes special care of this by synchronizing the threads passing the data from the OpenStack agent on to the fabric. Thus when a new deployment request comes down for execution, the software is notified and stops receiving new transfers from that point on, while at the same time locking the deployment thread until all outstanding transactions have completed. The VM image (essentially the bitstream) can then be passed on to DevC driver which will perform the reconfiguration. The locks will then be released so traffic can start flowing again to and from the accelerator.

Finally, the software stack includes a monitoring component which reads the monitoring data of the dedicated AXI DMA channel, writes them into a local file, and transmits them to a remote server over the well-known curl utility, which has been cross compiled for the Zynq platform.

8.3.3 Making OpenStack FPGA-Aware

When it comes to deploying a VM using a multi-tenant OpenStack topology, the OpenStack compute worker node is only one side of the coin. The other side is the OpenStack controller node which is responsible for running services that direct and manage the deployment of tasks in the pool of worker nodes it supervises. It also provides a single point of access through API services for communication with the other components of OpenStack. Deploying an HW VM correctly requires not only an adapted worker but also and an FPGA-aware OpenStack Controller node. Related aspects of the controller's functionality that were taken into account and modified in order to make it FPGA-aware for the needs of the proposed solution are listed below:

- Adaptation of the scheduler service to recognize, locate, and manage programmable logic-based devices and assign HW VMs to them. The allocation of VMs to tenants is done based on several filtering functions that apply criteria to select the most suitable tenant that will host a VM. The selection is done based on a predefined

set of metrics and information gathered from the monitoring services running on the worker nodes.

- The OpenStack Nova API service has been extended in order to include the calls to platform-specific functions. In this direction, additional commands have been added to the API that allows the management of the HW VM images by identifying and deploying them. So far the changes are command-line only. It goes without saying the extensions on the Nova API are backward compatible and are implemented with respect to the Nova developers policies. This is accomplished by taking advantage of the API Microversions framework that allows for specification of the API version that will be used in any circumstance.
- The Conductor service that also runs on the Controller node had to be slightly adapted in order to maintain access to the default relational database used by OpenStack for storing stateful information regarding the status of the platform. OpenStack's messaging queue is put to use to an extent that allows the brokerage of our system-specific messages between worker nodes, API service, scheduler, and the network service described below.

Another important aspect of deploying VMs over OpenStack is the provision and management of basic networking services. For the allocation of IP addresses and setup and configuration of the virtual networks of the host nodes (PCPs), a Nova-network service runs on the controller node that provisions services such as NAT and DHCP. A Nova-network client that executes on the worker node is responsible for configuring the network interface of the respective PCP-host. The Nova-network service was selected over the more advanced OpenStack Network Service (neutron) due to the added complexity and due to the fact that our requirements at this stage of the platform do not command for advanced networking topologies, or services such as load balancers and VPN which are offered by neutron.

Since the PCP is a compute only node, our modifications focused on the Nova component which is responsible for managing compute tasks but also extend to Glance, which administers the VM images used to deploy the VMs. Thus, the Glance component was modified to be able to manage HW VMs. Glance can now store these VMs and identify which ones are destined for FPGA SoCs and which not, avoiding mishaps during deployment.

It's worth noting that the changes performed are generic and do not limit PCP to being used with the development system currently used. Instead, any FPGA SoC can be used and even any FPGA provided its connected to a processor which can act as proxy for the OpenStack worker.

8.4 Experimental Results

PCP has been implemented using a Xilinx ZC706 development board which contains a Zynq xc7z045 FPGA SoC device. The board contains a Gigabit Ethernet interface which is used to connect the CPUs to the OpenStack controller and the traffic source and sink and an SD card which contains the boot image for the A9 CPUs. Thus, net-

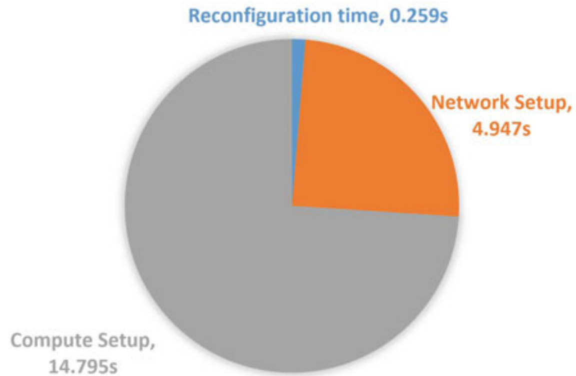

Fig. 8.2 Distribution of time lost during the deployment of an HW virtual machine

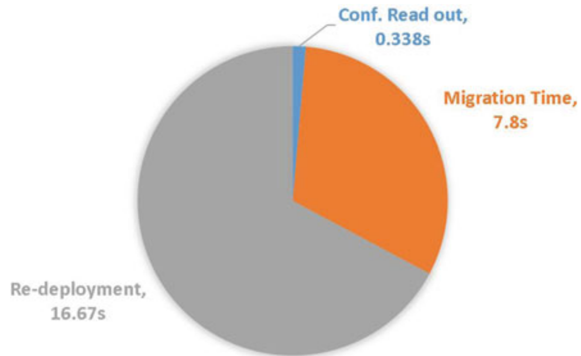

Fig. 8.3 Distribution of time required to migrate a virtual machine from one FPGA SoC node to another

work traffic entering the system is fed directly into the CPU for processing meaning all traffic to and from the accelerator goes through the A9s. This is mandatory since OpenStack has to perform network address translation on incoming data, which is performed at the SW level.

Evaluating the performance of our platform entails two separate aspects. First of all, we must ensure that the software stack running on the relatively weak A9 processors doesn't limit the capabilities of the programmable logic meaning that all OpenStack-related functionality is performed in a timely fashion. Furthermore, it is equally important to show that the HW VM running on the system's programmable fabric can indeed provide tangible benefits for a common application (Fig.8.5).

Another important aspect of VM deployment is migrating a VM from one node to another. This process was described in Sect. 8.3.2 and entails reading out the HW from the programmable logic via the DevC driver, sending it to the OpenStack controller and then redeploying it to a new device. We have measured the time the whole process takes and provided the data, breaking it down into its constituent parts (Fig. 8.2). Figure 8.3 illustrates the results averaged out over a set of 100 measurements.

The figure shows that reading the configuration out of the existing deployment (including stopping all traffic to that VM first) takes up only a small fraction of

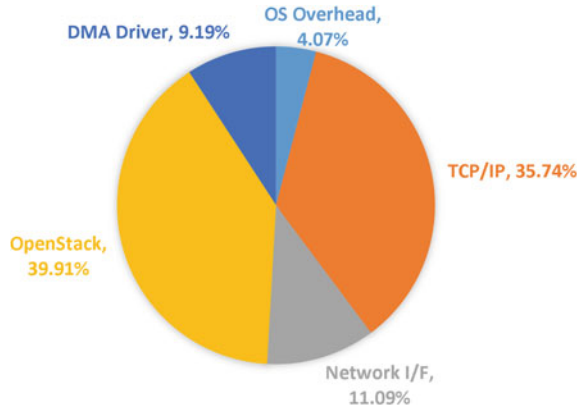

Fig. 8.4 Distribution of overhead when transmitting data through the SW stack

the total time it takes to migrate the VM completely. Migration time stands for the time required to send the data back to the OpenStack controller, while redeployment involves all the tasks detailed in Fig. 8.3, which are repeated when the VM is passed on to a new FPGA SoC node.

In order to evaluate the overhead caused by OpenStack processing on the Zynq's PS subsystem, we measure the time it takes to deploy one HW VM from the moment the appropriate command is passed to the controller, until the completion of the partial reconfiguration on the programmable fabric. Figure 8.3 illustrates how much time each part of the deployment cycle takes. The total process takes almost exactly 20 s with the bulk of the time being consumed by the setup of the compute instance followed closely by the creation of the necessary virtual network interfaces. While not negligible this time is on par with the deployment of SW VMs especially when one considers that an HW VM is readily available after being instantiated whereas an SW VM also has a substantial boot time. The actual reconfiguration of the fabric itself takes only a fraction of a second and can be safely ignored in this context.

The second facet that needs to be evaluated is whether the overhead incurred on the data when passing through the OpenStack worker during the HW VM's operation is low enough as well as the identification of sources of delay so that future improvements may be targeted accordingly. Figure 8.4 provides an overview of the form of a normalized pie chart of where in the software stack the time is spent when sending and receiving data traffic from and to the HW VM.

It's obvious from the above that the bulk of the processing cycles goes into TCP and OpenStack processing. TCP/IP is well known to be a processor cycle-hog [6], and this is verified on this platform as well even when only using a 1Gbps link. It is clear that the current software would not be able to support higher network data rates unless some sort of TCP acceleration (e.g., [12]) was utilized. This however would be implemented in the programmable logic and thus siphon resources away from the dynamic area and would still require the data to be shipped to the processor for the required OpenStack processing. OpenStack itself performs address translation on all incoming and outgoing packets, which is the reason behind its high cycle use.

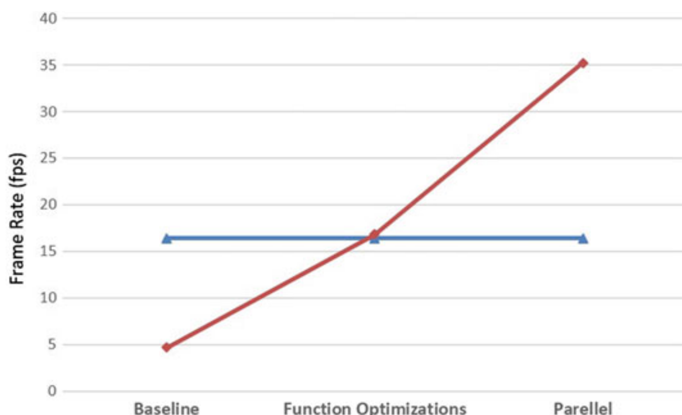

Fig. 8.5 Performance comparison of various implementations of the h264 decoder with a Xeon E5-2637 for a VGA video

A straightforward solution would be to simplify this processing given that PCP is a single-tenant platform, and thus, cohabitation of multiple user VMs is precluded. This is consigned to future work.

For the evaluation of the PCP, we have developed an h264 video decoder based on the OpenH264 software code provided by Cisco [5]. That code was reworked extensively to convert it to synthesizable C++ code which was then fed into Vivado HLS. There are three different versions of the design as shown in Fig. 8.5. The leftmost version represents the initial synthesizable version with no additional optimizations applied. The changes in this version can be essentially summed up in eliminating dynamic memory management prevalent in many areas of the original code. Thus the buffers and the context for each NAL [14] are now statically allocated.

The middle version includes optimizations made to the function hierarchy to minimize the impact of function calls, while the rightmost version represents a version that has been optimized and parallelized until the whole of the dynamic area was used up. It's worth noting that despite these modifications, the code was never rewritten from the ground up, and thus, it still maintains a structure which is not necessarily optimal for an HW module. Nevertheless, it shows that even with relatively modest reworking of existing SW code, considerable benefits can be reaped with the PCP.

Besides performance, resource use is an important parameter in FPGA designs. Table 8.1 provides the resource use numbers for the three variations of our h264 decoder. We can see that, as expected, the higher performing, more optimized variants consume significantly higher resources with the parallel implementation occupying a sizable chunk of the Zynq 7045 device used. Take in correlation with the performance number in Fig. 8.5 we can see that attaining a significant boost in comparison to standard x86 CPUs requires that a considerable part of the whole device is used (keep in mind that the number quoted on Table 8.1 does not include the overhead for the static area of our design, which is roughly 4.5% of the resources, as stipulated in

Table 8.1 h264 Decoder resource use on a Zynq 7045 FPGA SoC

	LUTs (%)	FFs (%)	BRAM (%)	DSP (%)
Baseline	31172/14.26	35151/8.04	19/3.57	36/3.96
Function optimization	71526/37.72	74062/16.94	36/6.67	78/8.7
Parallel	151381/69.25	148561/33.98	85/15.63	156/17.34
Total (Zynq 7045)	218600/100	437200/100	545/100	900/100

Sect. 8.3.2). This vindicates our decision to keep the dynamic area single-tenant in order to simplify the design of the system.

8.5 Conclusions

This paper introduced the programmable cloud platform, an FPGA SoC-based system which allows programmable logic to be exposed as a resource over a standard cloud environment like OpenStack. The system utilizes the devices ARM A9 processors to run the OpenStack agent and thus frees the programmable logic to accelerate tasks which are deployed as HW VMs. We use an application amenable to FPGA acceleration like h264 decoding to show that the platform allows for the swift deployment of the VNF and can feed it with sufficient data to reap tangible performance benefits. However, the cost of running OpenStack on a relatively weak processor is significant, and thus, one of the key issues to be tackled in the immediate future is the streamlining of the SW part to minimize this overhead. Additional work includes the extension of the platform's capabilities by integrating additional interfaces into the dynamic area.

Acknowledgements This work was undertaken under the EU FP7 Information Communications Technologies T-NOVA project, which is partially funded by the European Commission under grant 619520.

References

1. Asiatici M, George N, Vipin K, Fahmy SA, Ienne P (2016) Designing a virtual runtime for fpga accelerators in the cloud. In: 2016 26th international conference on field programmable logic and applications (FPL), pp 1–2
2. Bobda C, Majer A, Ahmadinia A, Haller T, Linarth A, Teich J (2005) The erlangen slot machine: increasing flexibility in FPGA-based reconfigurable platforms. In: Proceedings 2005 IEEE international conference on field-programmable technology, pp 37–42
3. Byma S, Steffan H, Bannazadeh G, Leon A, Chow P (2013) FPGAs in the cloud: booting virtualized hardware accelerators with openstack. In: 2014 IEEE 22nd annual international symposium on field-programmable custom computing Machines (FCCM), May 2013

4. Chen F, Shan Y, Zhang Y, Wang HF, Chang X (2014) Enabling FPGAs in the cloud. In: Proceedings of the 11th ACM conference on computing frontiers, May 2014
5. Cisco Inc. http://www.openh264.org
6. Foong AP, Huff TR, Hum HH, Patwardhan JR, Regnier GJ (2003) TCP performance re-visited. In: Proceedings of the 2003 IEEE international symposium on performance analysis of systems and software, ISPASS '03. IEEE Computer Society, Washington, DC, USA, pp 70–79
7. Ge X, Liu Y, Du DH, Zhang L, Guan H, Chen J, Zhao Y, Hu X (2014) OpenANFV: accelerating network function virtualization with a consolidated framework in openstack. In: Proceedings of the 2014 ACM conference on SIGCOMM, Aug 2014
8. Jouppi NP, Young C, Patil N, Patterson D, Agrawal G, Bajwa R, Bates S, Bhatia S, Boden N, Borchers A, Boyle R, luc Cantin P, Chao C, Clark C, Coriell J, Daley M, Dau M, Dean J, Gelb B, Ghaemmaghami T, Gottipati R, Gulland W, Hagmann R, Ho CR, Hogberg D, Hu J, Hundt R, Hurt D, Ibarz J, Jaffey A, Jaworski A, Kaplan A, Khaitan H, Koch A, Kumar N, Lacy S, Laudon J, Law J, Le D, Leary C, Liu Z, Lucke K, Lundin A, MacKean G, Maggiore A, Mahony M, Miller K, Nagarajan R, Narayanaswami R, Ni R, Nix K, Norrie T, Omernick M, Penukonda N, Phelps A, Ross J, Ross M, Salek A, Samadiani E, Severn C, Sizikov G, Snelham M, Souter J, Steinberg D, Swing A, Tan M, Thorson G, Tian B, Toma H, Tuttle E, Vasudevan V, Walter R, Wang W, Wilcox E, Yoon DH (2012) In-datacenter performance analysis of a tensor processing unit. In: To appear at the 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada, June 24–28
9. Karras K, Manolakos ES (2008) An embedded dynamically self-reconfigurable master-slaves mpsoc architecture. In: 2008 international conference on field programmable logic and applications, pp 431–434
10. Pham KD, A Jain, Cui J, Fahmy S, Maskell DL (2013) Microkernel hypervisor for a hybrid arm-fpga platform. In: 2013 IEEE 24th international conference on application-specific systems, architectures and processors (ASAP), pp 219–226
11. Scott Sirowy AF (2008) Wheres the beef? why FPGAs are so fast. In: Microsoft Technical Report - MSR-TR-2008-130
12. Sidler D, Alonso G, Blott M, Karras K, Vissers K, Carley R (2015) Scalable 10 gbps TCP/IP stack architecture for reconfigurable hardware. In: 2015 IEEE 23rd annual international symposium on field-programmable custom computing machines (FCCM), pp 36–43
13. Wang K, Angstadt K, Bo C, Brunelle N, Sadredini E, Tracy T, Wadden J, Stan M, Skadron K (2016) An overview of micron's automata processor. In: 2016 international conference on hardware/software codesign and system synthesis (CODES + ISSS), pp 1–3
14. Wiegand T, Sullivan GJ, Bjontegaard G, Luthra A (2003) Overview of the h.264/avc video coding standard. IEEE Trans Circ Sys Video Technol 13(7):560–576. http://www.rsc.org/dose/title-of-subordinate-document Cited 15 Jan 1999
15. Xilinx Ultrascale+ MPSoC. https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html