Virtualized FPGA Accelerators for Efficient Cloud Computing

Suhaib A. Fahmy*, Kizheppatt Vipin†, Shanker Shreejith‡
*School of Engineering, University of Warwick, United Kingdom
†School of Engineering Sciences, Mahindra École Centrale, Hyderabad, India
‡School of Computer Engineering, Nanyang Technological University, Singapore
Email: sfahmy@ieee.org

Abstract—**Hardware accelerators implement custom architectures to significantly speed up computations in a wide range of domains. As performance scaling in server-class CPUs slows, we propose the integration of hardware accelerators in the cloud as a way to maintain a positive performance trend. Field programmable gate arrays (FPGAs) represent the ideal way to integrate accelerators in the cloud, since they can be reprogrammed as needs change and allow multiple accelerators to share optimised communication infrastructure. We discuss a framework that integrates reconfigurable accelerators in a standard server with virtualised resource management and communication. We then present a case study that quantifies the efficiency benefits and break-even point for integrating FPGAs in the cloud.**

I. Introduction

Cloud Computing promises shared elastic access to unlimited compute resources in a similar view to traditional utilities like the power grid. Virtualisation enables efficient scaling and sharing as user needs change. Established cloud service models offer access to virtualised hardware, to tuned application design platforms, or simply to self-contained application software hosted in the cloud. A key driver behind the increase in demand for cloud computing has been the exponential increase in use of mobile computing devices that often lack the computational power to complete complex tasks such as voice recognition. Computing in the cloud benefits from server-grade CPUs with advanced computational datapaths, large caches, and multiple cores.

While the flexibility, scalability, and affordability of cloud computing are well established, performance and efficiency remain matters of concern. By virtue of the virtualisation of resources, the distributed communication among sometimes disparate nodes, and fluctuations in demand, running complex applications can be challenging [1]. Furthermore, the performance of server CPUs is not scaling at the rates previously observed [2]. Poor computational performance due to the overheads of virtualisation can also severely impact response time and hence user experience, even for simpler tasks. Efficiency is also becoming a major concern as datacenters start to consume a noticeable proportion of the world's energy budget.

One strategy discussed for overcoming stalled performance scaling is to incorporate heterogeneous resources better suited to complex computation [3]. GPUs can offer significant performance benefits but cloud integration can be troublesome, since the architectures are designed to be used monolithically [4]. Hence, they are generally offered as a fixed resource using the Infrastructure-as-a-Service (IaaS) model, leading to potential under-utilisation and over-provision. GPUs are also power-hungry, and hence, do not contribute significantly to improving energy efficiency.

FPGAs have also been explored more recently. They can offer significant speed-up in execution of a wide variety of tasks and are also significantly more power-efficient [5], [6]. Recent developments are easing the integration of FPGAs in the datacenter. At the server hardware level, IBM's POWER8 Coherent Accelerator Processor Interface (CAPI) [2] allows tighter coupling between the processor and a co-processing peripheral. Intel's XEON+FPGA integrates an FPGA with a XEON processor in a single chip package, and their recent purchase of Altera underlines the importance of hardware accelerators in the datacenter. Microsoft recently presented a comprehensive demonstration of the benefits of FPGAs in the datacenter applied to the Bing search algorithm [7].

These initial proofs of concept make the case for a more serious investigation of FPGAs as a general cloud computing resource. So far, however, FPGAs have only been used as static accelerators, designed once and used for a single function. In the cloud context, the ability to modify accelerator functions at runtime in a multi-user environment is essential. This requires new techniques in hardware design, interfacing, accelerator management, OS integration, and programming models. In this paper, we present some background on FPGAs, then detail a platform for integrating virtualised accelerators on FPGAs, and finally, present a case study that quantifies potential benefits.

II. Background

A. Field Programmable Gate Arrays

Field programmable gate arrays (FPGAs) are commercial off-the-shelf silicon devices that can be programmed to implement custom digital circuits. Fundamentally, they consist of a mesh of basic circuit resources that can be combined to create complex architectures. The key performance and power benefits are realised by designing custom computational datapaths suited to a particular application. An architecture is described in a hardware description language (HDL), or more recently using high level algorithmic code, and automated tools work out how to build it using the components in the

FPGA, how they should be arranged on the grid of the FPGA and connected, finally generating a bitstream — a binary file loaded into the FPGA to implement the circuit.

By exploiting paralellism in an algorithm and tailoring the computational datapath(s), an FPGA accelerator offers a significant performance improvement over software on a processor, often consuming significantly less power. Compared to ASICs, there is no manufacturing process, turnaround time is significantly reduced, and the design can be changed at any time simply by loading a new bitstream.

Within the cloud context, FPGAs represent a promising accelerator platform due to this application flexibility that preserves the generality of the resource. Furthermore, an FPGA can be shared spatially between distinct, isolated accelerators that can be reprogrammable at runtime. This allows for a more fine-grained approach to using hardware resources that fits the ideas of scalability and sharing that are so fundamental in the cloud.

B. Coupling FPGAs with Processors

The reconfigurable computing community has explored the coupling of FPGAs with general purpose processors for a long time [8]. Traditionally, the FPGA has acted as a standalone accelerator with communication through I/O interfaces, resulting in high communication overheads. Over time, the FPGA has been brought closer to the CPU, resulting in higher communication bandwidth and lower latency [9]. More recently, integrated systems on chip, combining both general purpose processors and FPGAs have emerged, such as the Xilinx Zynq, offering extremely high data bandwidth between the processor and FPGA, allowing for high performance applications with interleaved hardware and software execution. Coupling is of great importance as slow communication can severely decimate the potential benefits of acceleration [10]. Recent FPGAs support very high bandwidth serial communication interfaces, including PCI Express (PCIe), allowing them to be integrated in standard computing infrastructure, with saturated interface bandwidth. Developments such as CAPI will enhance this coupling further.

C. Partial Reconfiguration

Apart from the reprogrammability of FPGAs, partial reconfiguration could be key to adoption in the cloud. Just as the virtual CPUs allocated to users are abstracted from the physical CPUs of a server, allowing scaling and sharing, hardware accelerators should also be virtualised on the hardware resources. Partial reconfiguration is where only a part of the FPGA is reconfigured instead of the whole device. Multiple accelerator slots can host independently-managed accelerators that do not interfere with each other, while the communication infrastructure remains in place throughout operation.

Partially reconfigurable regions (PRRs) are defined, as shown in Fig. 1, each able to host different accelerators. Partial bitstreams contain all the configuration information required to place an accelerator into a PRR. The static region is the part of the FPGA that is not reconfigured at runtime and contains all the communication interfaces and reconfiguration management circuitry. Partial reconfiguration also allows for much faster reconfiguration of accelerators.

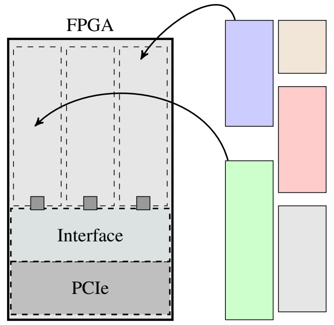

Fig. 1. Partial reconfiguration of an FPGA to allow separate accelerators to be loaded into distinct slots.

III. Existing Work

Incorporation of FPGAs in the datacenter is a relatively new area of research. One proposed application is to provide better security and privacy by offloading sensitive data processing into hardware since possible attack vectors are limited [11]. Netezza's data warehousing appliances perform complex data filtering on FPGAs, with additional compression/decompression performed by spare resources [12]. Microsoft's Catapult architecture represents the first detailed investigation of applying FPGAs within an enterprise-level datacenter application [7]. The document ranking part of Bing search is accelerated using hardware split across 8 FPGAs within a rack. They report almost doubled throughput in search ranking at a cost of only 10% increased power consumption and 30% increased total cost of ownership. Baidu presented accelerated neural networks on FPGAs in the datacenter offering an order of magnitude better performance at minimal additional power cost [13]. In [14], the authors present FPGAs in the cloud as standalone resources, treated separately from standard (server) resources. In this setup the whole application must run in the FPGA and traditional FPGA reconfiguration over JTAG means extra cabling is required and reconfiguration is slow. FPGAs were also explored for implementation of core functions in intelligent personal assistants (IPAs), with their performance per Watt exceeding CPUs and GPUs by a significant margin [15].

Open source interface frameworks have emerged allowing FPGAs to be integrated in PCs with communication throughput between the host and FPGA close to the limits of modern PCIe interfaces [16]. Reconfiguration and communication over a single PCIe interface has also been demonstrated [17], enabling accelerator swapping and overcoming the need for extra cabling and drivers which can be problematic in a tightly managed datacenter environment.

Key challenges to be addressed for a cloud-centric integration of FPGAs are:

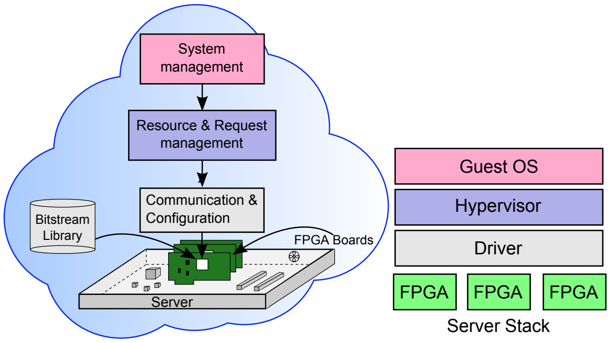

Fig. 2. Proposed FPGA in the cloud architecture.

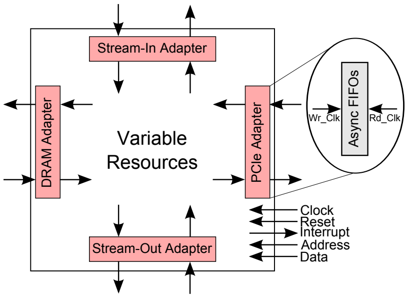

Fig. 3. A vFPGA (PRR) showing interface signals and adapters.

- Support for dynamically reconfigurable accelerators to support changing application needs with low latency
- Maximising communication throughput to multiple accelerators with fair, segregated sharing
- Maximised usage of FPGA resources at all times through efficient scheduling and allocation
- Easy integration of accelerated tasks within software applications

The work we have seen to date focusses primarily on FPGAs as static accelerators or as monolithic devices brought online for a particular task. We explore the benefits of virtualised dynamically reconfigurable accelerators, with a view to using FPGAs as a general computing resource alongside standard resources.

IV. Cloud FPGA Framework

Our framework integrates a PCIe based FPGA board into a standard datacenter server. The FPGA is partitioned into separate accelerator slots. The PCIe interface manages reconfiguration of the accelerators and movement of data into and out of them. Accelerator functions are either stored in a library on the host machine as partial bitstreams or can be uploaded by the user.

We have extended a previous open-source partially reconfigurable FPGA test platform [17] to support multiple independent accelerators and added the required software framework to enable accelerator management. The communication interface is implemented in the FPGA static logic and can manage multiple accelerators concurrently. A built-in arbiter guarantees fair communication bandwidth to every accelerator when multiple accelerators are communicating with the host server. Accelerators are also configured over PCIe, providing superior reconfiguration performance and avoiding the need for external cabling.

We have developed software infrastructure to manage low-level communication through a driver and high-level virtualisation through a hypervisor. When an accelerator is to be configured in the FPGA, the hypervisor decides on the optimal PRR to host it and initiates reconfiguration. The hypervisor also maintains a list of PRRs and configured accelerators to avoid unnecessary reconfiguration when a required accelerator is already present in the FPGA and not in use. An outline of our platform is depicted in Fig. 2.

A. Hardware Infrastructure

To host FPGAs in the cloud, standard datacenter servers are used with commercially available FPGA development boards.

1) Host Servers: The server CPUs run the software components for user request, FPGA resource, and communication management. The servers also maintain a database of hardware accelerators in the form of a bitstream library which can be augmented with custom user designs. Servers can host multiple FPGA boards of varying logic capacity and performance. Using off-the shelf FPGA boards offers easy upgradability and ensures minimal cost. These FPGA boards are interfaced through a PCIe Gen 3×8 interface and come with on-board DRAM for FPGA off-chip storage.

Each physical FPGA is divided into multiple partially reconfigurable regions (PRRs), which act as virtual FPGAs (vFPGAs) for hosting accelerators. vFPGAs are interfaced with the host FPGA's PCIe and DRAM physical interfaces for communication and data storage. The logic to manage these physical interfaces is implemented in the FPGA *static logic*, meaning it is not reconfigured during FPGA uptime. An interface switch ensures each vFPGA is served in a fair manner with round robin arbitration for access to the PCIe and DRAM data streams. vFPGAs can also be prioritised with higher bandwidth if application needs are different. The switch also implements independent DMA controllers to manage data transfer between each vFPGA and the server. Each physical FPGA has a reconfiguration controller, which enables partial reconfiguration of the vFPGAs using a dedicated DMA controller for high-speed partial bitstream transfer from the server.

2) Virtual FPGAs (vFPGAs): A vFPGA is the smallest *FPGA instance* available to cloud users, and the size can be set based on anticipated demand, but also changed through a full reconfiguration of the FPGA. A vFPGA can be configured with a compatible partial bitstream to implement a virtual FPGA accelerator (vFA). A vFPGA can host multiple vFAs, whose size in terms of FPGA resources is smaller than that of the vFPGA. To enable portability and simplify vFA design, the vFPGAs all have a standard interface: a single AXI4-Stream interface to the PCIe core and another AXI4-Stream interface to external DRAM. Two other stream interfaces connect adjacent vFPGAs, enabling accelerator chaining. Each vFPGA also has an address/data interface accessible over PCIe, an interrupt interface, and clock and reset signals, all used to manage the functioning of the vFAs it hosts.

Every stream interface to/from a vFPGA is integrated with the rest of the FPGA fabric through *adapters* as shown in Fig. 3. These adapters are essentially AXI4-Stream based asynchronous FIFOs, which enable the logic within the vFPGA (vFAs) to run at a different clock frequency from the interface. This is important as some vFAs may not be able to run at the interface line rate.

When multiple vFPGAs are active simultaneously, it is important to make sure that the communication bandwidth between the host and the vFPGAs is fairly partitioned. Lack of bandwidth management leads to hosted vFAs suffering from data starvation, leading to performance degradation. A hardware arbitrator in the FPGA ensures bandwidth is equally partitioned between all the active vFPGAs using modified round-robin arbitration that considers only active vFPGAs. This allocation is completely abstracted from the vFPGAs ensuring a clean partitioning between them. Similarly, in the host driver, separate DMA buffers are reserved for each vFPGA to ensure data isolation.

B. Software Infrastructure

Managing hardware resources requires a number of software components tailored to this task.

1) FPGA Driver: This is responsible for managing low level PCIe input/output operations, DMA buffer management and interrupt management. Our driver implementation maintains separate buffers and interrupt queues to manage data transfer to each vFPGA enabling concurrent, partitioned DMA operations to multiple vFPGAs.

2) Application Programming Interface (API): A supplied API enables users to easily integrate vFAs in their software. API functions enable data transfer between the host server and vFAs, between off-chip DRAM and vFAs, vFA interrupt management, and vFPGA reconfiguration. To initiate a data transfer, the source and destination are passed as arguments to the relevant API function. The framework, with the help of the driver, configures the appropriate DMA controller in the FPGA to initiate the operation. Each data transfer is synchronised based on interrupt signals from the corresponding DMA controller.

The public reconfiguration API only allows access to vFAs stored in the accelerator library. However, a private reconfiguration API can be exposed to users to support implementation of their own vFAs subject to security considerations.

3) Hypervisor: The hypervisor is implemented as part of the cloud virtualisation layer, supporting management of FPGA resources.

Resource Management: When the cloud supports multiple vFPGAs and multiple FPGA boards, a *resource manager* is required to manage them. The allocation of vFAs to vFPGAs is hidden from the user in the same way that they would not know which physical CPU core a vCPU is run on. This enables the management infrastructure to consolidate resources for best performance and efficiency. We implement a resource manager that considers the correlation between vFA size and vFPGA size. The resource manager maintains a list of available vFPGAs, which accelerators they each support, and their allocation status. It also has access to the accelerator database, where partial bitstreams corresponding to different vFAs targeting multiple vFPGAs are stored.

When a user requests an accelerator, a new software thread is generated to serve it. It requests a specific vFA reconfiguration and later manages data movement between the host and the vFA. When the resource manager receives a vFA reconfiguration request, it selects the smallest vFPGA capable of hosting it from the pool of free vFPGAs. The hypervisor makes sure that only a single vFPGA is reconfigured at any time, since partial reconfiguration is not preemptive. Once a vFPGA is selected, it is marked as *busy* in the vFPGA list and the configured accelerator is also noted. If a vFPGA cannot be allocated, the user request is rejected and the corresponding software thread is destroyed. A vFPGA is returned to the free pool once the request has been serviced. Once freed, a subsequent request for the same accelerator can be serviced without reconfiguring. Unused vFPGAs can also be configured with *blank* partial bitstreams to reduce power consumption.

If the resource manager is unable to find a free vFPGA to host the accelerator, the user request is rejected, and the request can be processed in software instead. By selecting the smallest possible vFPGA for the accelerator, the resource manager tries to maximise the number of vFPGAs available for implementing larger accelerators, thus minimising request rejections. Presently, vFPGA allocations are non-preemptive and a vFPGA is returned to the free pool only after fully servicing a user request. It is clear that vFPGA granularity must be chosen to reflect the types of accelerators and expected user workloads for benefits to be maximised.

Security Management: Using FPGAs in a shared infrastructure environment can create security issues, since users with the ability to reconfigure FPGAs might launch malicious attacks through corrupt bitstreams. This can be circumvented either through abstracting the accelerator library so no direct accelerator configuration is allowed, or through extra security steps during the client accelerator design process. Features such as *bitstream encryption* and *CRC insertion* are supported by FPGA implementation tools and bitstream authentication and error checking are supported in the FPGA silicon.

We implement additional bitstream authentication to prevent potential malicious bitstreams using *bitstream watermarking*. In the server, every new bitstream is authenticated based on the watermark before being used for reconfiguration and implementation scripts are secured to prevent tampering.

4) Middleware: This is the software running on the client machine which enables it to access the cloud services. We have developed a prototype lightweight middleware interface that enables users to send accelerator requests (or custom vFA bitstreams) and container software using the cloud API, then

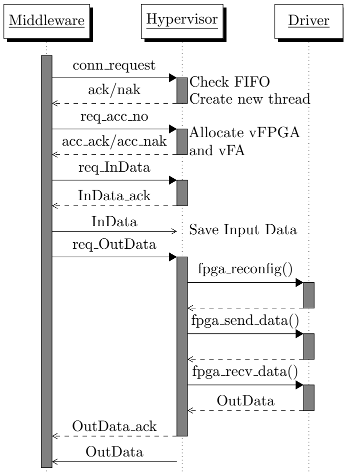

Fig. 4. Middleware communication protocol.

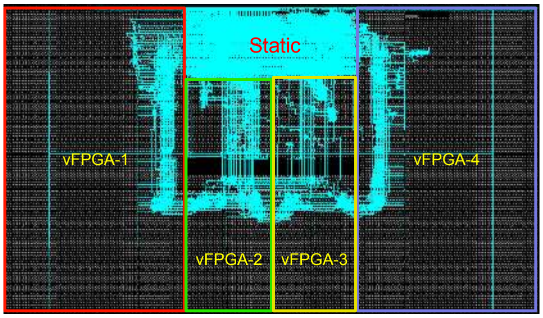

Fig. 5. Virtex-7 floorplan showing 4 vFPGAs.

send input data to the cloud using the server's IP address, before receiving the resulting processed data. The middleware uses Linux sockets with TCP/IP to access the server.

Fig. 4 shows our middleware protocol for accessing library accelerators. Users initially request the cloud service by sending a *connection request* to the cloud server. If the request is accepted, the hypervisor pushes the request onto a FIFO and acknowledges it. When the client receives an acknowledgement, it can request a specific vFA using a predefined accelerator tag. The hypervisor resource manager tries to allocate a vFPGA for the vFA; if one is allocated, an acknowledgement is returned and the client sends the input data. The software thread on the server reconfigures the vFPGA with the requested vFA, sends it the input data, and receives the processed data, which is then sent back to the client along with a data request acknowledgement. This is currently mostly a proof of concept, and we are developing more advanced features.

V. Case Study

We have implemented the proposed cloud hardware infrastructure on a Xilinx VC709 FPGA board containing a XC7VX690T FPGA, supporting PCIe Gen 3×8, and hosting 8GB of on-board memory. It is hosted in an HP Z420 with an Intel Xeon E5-1650 v2 CPU running at 3.5 GHz with 16GB of RAM. The large FPGA offers ample resources for complex accelerators. Our custom hardware logic for PCIe and DRAM communication and reconfiguration management consumes about 7% of the FPGA area and is implemented statically. The remainder of the FPGA is divided into 4 vFPGAs as shown in Fig. 5, allowing this implementation to host up to 4 independent accelerators concurrently. A new vFA can be configured in under 16 ms.

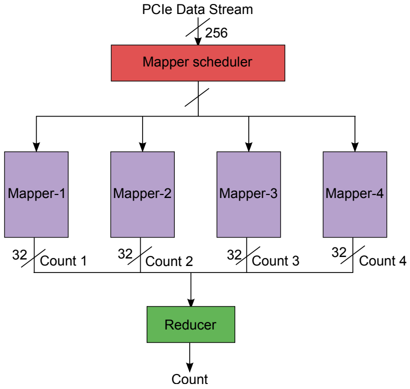

Fig. 6. vFA architecture for word count.

As an application case study, we built a map-reduce accelerator for word counting, which finds the number of occurrences of a specified word in a large data set, useful in data mining applications. The vFA receives a 256-bit data stream at 250 MHz over PCIe. A *mapper scheduler* passes the data to one of a number of *mappers*, that each count the number of occurrence of the specified word. Data is stored in a 256-bit to 8-bit asymmetric FIFO which buffers data locally and splits it into distinct characters that are sequentially moved into a shift register. When a non alpha-numeric character is encountered, it is compared with the query word and if they match, an internal counter is incremented and the shift register is flushed. The accelerator design is highly scalable supporting a large number of parallel mappers in a single vFA. We built multiple vFAs targeting the same vFPGA with 1–64 mappers. The outputs of all the mappers are connected to a single reducer, which periodically adds the output from all the mappers and finds the total count. This value can be read-back through the PCIe interface. A software version of the application was implemented in C based on the efficient Boyer-Moore algorithm [18].

Software performance saturates at 8 mappers as the server processor cannot efficiently support more threads, resulting in a peak throughput of 40.8 MB/s. A single hardware mapper provides up to 240 MB/s throughput and as the number of mappers increases, the performance improves up to over 6.8 GB/s for 32 mappers, beyond which there is no significant improvement as the PCIe bandwidth is saturated (86%). All experiments start with a file of given size buffered in memory and the results being stored to memory, so the hardware results include all required movement of data to and from the FPGA. This performance gap of 166× varies by application, but over a full order of magnitude is possible for many classes of applications.

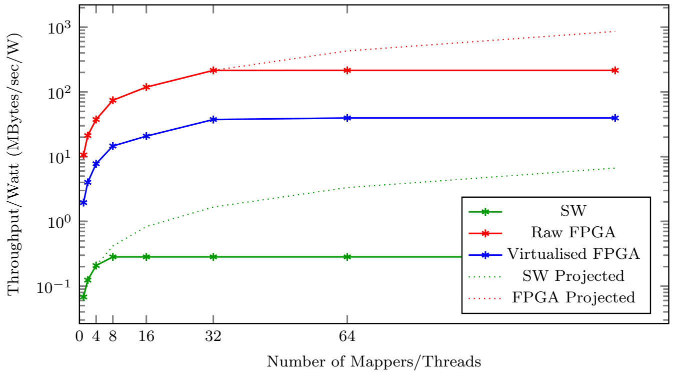

Fig. 7. Performance per Watt with varying number of mappers.

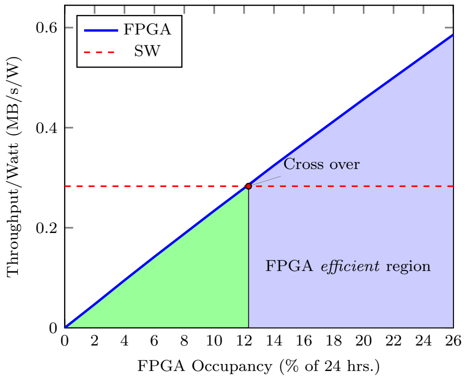

Fig. 8. Break even point for the FPGA to be efficient.

We also investigated performance per Watt and quantified the cost of virtualising the FPGA with distinct slots for separate mappers and separate management of the vFAs. The standalone idle power consumption of the target server is about 75 W. The VC709 board consumes 24 W, with negligible change when running accelerators. The software application consumes an extra 70 W (145 W total) when at peak throughput. The vFPGA resource management threads result in a total CPU power consumption of 128 W at peak throughput. Fig. 7 shows the significant gap between hardware and software, and shows that even factoring in the software management of the vFAs and the reconfiguration time to load them, the efficiency of the virtualised hardware (blue line) remains 2 orders of magnitude higher.

Adding an FPGA board adds a fixed power consumption overhead to the server, so we consider how much usage the FPGA requires before this overhead is amortised. Fig. 8 shows that the virtualised FPGA implementation surpasses the software only computational efficiency once the FPGA is used over 12% of the time.

VI. Conclusion

Integration of heterogeneous hardware resources in the cloud offers an opportunity to significantly improve performance and computational efficiency, while overcoming stalled CPU performance scaling. FPGAs offer the advantages of high communication bandwidth shared among multiple accelerators and dynamic loading of accelerators at run time through partial reconfiguration.

We have presented a prototype framework for integrating virtualised FPGA accelerators in the cloud using partial reconfiguration and virtualised communication interfaces. We also detailed a case-study that demonstrates that even with the virtualisation overhead, FPGAs offers a significant improvement in computational efficiency over software.

References

[1] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, "Performance analysis of high performance computing applications on the Amazon Web Services cloud," in *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2010, pp. 159–168.

[2] J. M. Tendler, "An introduction to the POWER8 processor," Presented at the IBM POWER User Group, Jan. 2014.

[3] G. Lee, B.-G. Chun, and R. H. Katz, "Heterogeneity-aware resource allocation and scheduling in the cloud," *Proceedings of HotCloud*, 2011.

[4] V. T. Ravi, M. Becchi, G. Agrawal, and S. Chakradhar, "Supporting GPU sharing in cloud environments with a transparent runtime consolidation framework," in *Proceedings of the International Symposium on High Performance Distributed Computing*, 2011, pp. 217–228.

[5] S. Kestur, J. D. Davis, and O. Williams, "BLAS comparison on FPGA, CPU and GPU," in *Proc. Int. Symp. VLSI (ISVLSI)*, 2010, pp. 288–293.

[6] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2009, pp. 126–131.

[7] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2014, pp. 13–24.

[8] T. J. Todman, G. A. Constantinides, S. J. Wilton, O. Mencer, W. Luk, and P. Y. Cheung, "Reconfigurable computing: Architectures and design methods," *IEE Proceedings–Computers and Digital Techniques*, vol. 152, no. 2, pp. 193–207, 2005.

[9] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "PipeRench: A reconfigurable architecture and compiler," *Computer*, vol. 33, no. 4, pp. 70–77, 2000.

[10] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq," *IEEE Embedded System Letters (ESL)*, vol. 6, no. 3, pp. 41–44, 2014.

[11] K. Eguro and R. Venkatesan, "FPGAs for trusted cloud computing," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 63–70.

[12] P. Francisco *et al.*, *The Netezza Data Appliance Architecture: A Platform for High Performance Data Warehousing and Analytics.* IBM Redbooks, 2011.

[13] J. Ouyang, S. Lin, W. Qi, Y. Wang, and B. Yu, "SDA: Software-defined accelerator for large-scale DNN systems," in *Proceedings of HOT CHIPS*, 2014.

[14] S. Byma, J. G. Steffan, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "FPGAs in the cloud: Booting virtualized hardware accelerators with OpenStack," in *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2014, pp. 109–116.

[15] J. Hauswald *et al.*, "Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

[16] M. Jacobsen and R. Kastner, "RIFFA 2.0: A reusable integration framework for FPGA accelerators," in *Proceeding of the International Conference on Field Programmable Logic and Applications (FPL) International Conference on Field-Programmable Logic*, 2013.

[17] K. Vipin and S. A. Fahmy, "DyRACT: A partial reconfiguration enabled accelerator and test platform," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2014.

[18] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762–772, 1977.