LynX: A Flexible FPGA Virtualization Framework for Heterogeneous Systems

Dario Korolija
ETH Zurich

Gustavo Alonso
ETH Zurich

Timothy Roscoe
ETH Zurich

Abstract

In the age of big data, with ever-increasing computing requirements, alternative approaches to generic CPU computing are a necessity. Since reconfigurable hardware has been shown to improve performance and energy efficiency, heterogeneous systems consisting of coupled CPU-FPGA architectures are gaining traction and are now available in several cloud based data centers. However, application development and flexibility of these systems is lacking as there is no underlying support for commons tasks and operations (similar to those an operating system provides on a CPU). We present LynX, a flexible shell for FPGAs providing various operating system-like interfaces and improving programming convenience whilst having a minimal impact on performance. LynX supports spatial-sharing and a multi-tenant environment with dynamic reconfiguration and isolation between concurrent applications. LynX employs a resource virtualization approach based on a unified virtual memory scheme bringing the FPGA and its local resources (memory, network) closer to the developer. Compared to existing proposals targeting similar platforms our framework offers significant advantages due to LynX's configurability and flexibility.

1 Introduction

Although Moore's law is slowing down [2], application demands continue to grow, leading to the use of alternatives to conventional CPUs. In particular, FPGAs are becoming pervasive accelerators, with performance and energy consumption close to that of an ASIC [7], without the associated high design and production costs, and with the flexibility to be reconfigured even at runtime [9, 10]. Like GPUs they provide a very high degree of parallelism, but FPGAs do so without restricting the computational model to dense computations and sequential access to instructions and data. As a result, FPGAs have been efficiently used in fields like signal and image processing [1, 3, 14], machine learning [5, 8, 15], in network data processing and database acceleration [4, 6, 13], to name a few.

However, FPGAs have drawbacks. Their clock frequencies are typically one order of magnitude lower than that of a CPU or GPU. Performance comes instead from deeply pipelined hardware designs with high concurrency and high degrees of specialization. The lack of proper software support and missing abstractions necessitate hardware expertise and experience from the developers and makes the usage highly unattractive for novice users. As FPGAs start to become parts of a larger ecosystem, being used in hybrid CPU-FPGA architectures rather than standalone, the added contrast between hardware and software development creates an additional problem. Hybrid CPU-FPGA systems require a layer facilitating interaction between FPGA and the CPU's OS. To date, such layers are primitive, often bound to concrete devices or brands, and lack the highly needed flexibility one would expect from a traditional operating system. Examples of the state of the art include Microsoft's Catapult [12] or Intel's HARP [11]. Both are focused on a specific group of applications and use cases, which in turn leads to missing features and resources. This emphasises the need to create new frameworks for these systems which will allow a more defined, coherent and abstracted interaction between software and hardware while at the same time trying to provide adaptability for various work loads.

We present LynX, aiming to bridge the gap between the software and hardware layers in modern heterogeneous systems. LynX offers flexibility with an operating system-like shell for FPGAs that is highly configurable and generic in the sense of being able to support a wide variety of applications and processing scenarios. LynX brings OS-like interfaces and functionality such as multitasking between multiple concurrent hardware processes, running in virtual FPGA regions, which can be dynamically reconfigured (resembling a context switch). On top of that we present a runtime manager which is able to dynamically schedule jobs in order to optimize their execution time. Our framework offers ease of use through the unified virtual memory scheme, focusing on local FPGA resources, which allows us to conceal the fragmentation through dynamic allocation of the memory and to expose a contiguous virtual address range to the users, greatly reducing the complexity of interacting with the FPGA from software. The dynamic allocation of FPGA on-board memory allows us to implement a custom access pattern leading to an increase in memory performance. We additionally integrate an RDMA network stack allowing remote memory operations, greatly expanding the possible use cases.

The key features of LynX, and its key contributions, are:

1. Flexbility of use and abstractions analogous to those found in operating systems.
2. Ease of use with unified virtual memory space between FPGA, its resources, and the host.
3. Support for basic system functionality and management of FPGA resources with dynamic memory allocation, demand paging and remote memory operations.

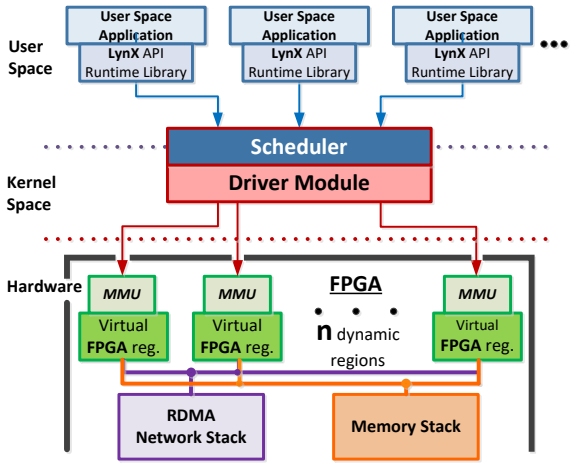

Figure 1. LynX top level framework overview

2 Design Goals and System Overview

The aim in hardware is to create a flexible shell able to accommodate a wide span of different application domains. Unlike with CPUs and GPUs, code running on one model of FPGA is not guaranteed to work on a different model. Moreover, in most existing FPGA shells, the features added to user's code are often fixed regardless of whether they are used, preventing the adaptation to different applications. Using LynX's common interfaces, programmers can quickly deploy both low level RTL and high level synthesis designs. Data can be streamed either from the host or the network. Additionally LynX also supports the GPU model where the data is first offloaded to local on-board memory which offers higher performance for iterative applications. Essentially all FPGA deployments today are single user, significantly reducing their potential, especially in the cloud. The flexible nature of LynX allows a single FPGA to run multiple applications at the same time through spatial and temporal sharing while managing and allowing each of those applications accesses to external resources (memory, network, host...). This permits a much broader usage of FPGAs and relieves programming complexity. In its current version, LynX supports a wide range of Xilinx FPGAs. Our goal is to add support for FPGAs from other vendors and interconnects in the near future.

Applications using a CPU and an FPGA are usually confronted with two separate, independent memory regions requiring explicit memory management greatly increasing the programming complexity. In contrast, the aim of LynX is to provide a unified address space between the FPGA, its resources, and the host. This decreases code complexity as the memory management, demand paging and the handling of all page faults is occurring in the background. Similar models have been proposed for GPUs, but besides the implementation on the FPGA, the difference of LynX is the ability to support multiple concurrent contexts in a provided multi-tenant environment. This unification into a single virtual address space greatly simplifies software-hardware interaction. The unified memory model is implemented through a custom memory management unit residing in the FPGA fabric which handles demand paging and supports multiple page sizes. This unit monitors accesses from the virtual FPGAs to both the host and to the local on-board FPGA memory. In LynX, the FPGA can access the complete user virtual address range and operate on pointer-rich data structures residing in memory either on the host or on the FPGA side. Dynamic allocation allows us to implement a custom striding memory access pattern to the on-board FPGA memory, enabling the optimization of performance when multiple DRAM channels are present, whilst preserving a simplistic common DRAM interface.

On top of our hardware layer, we implement a dynamic runtime manager with the goal of scheduling user jobs to the available virtual FPGA regions whilst minimizing the amount of costly operator exchanges via partial reconfiguration. Additionally, a hierarchical application interface exposed by the runtime manager allows the abstraction and minimization of interaction between low level FPGA communication and the end user. In software, the goal of LynX is to provide interfaces which permit quick integration of applications for the end user whilst hiding the complexity of the lower levels of the stack.

3 Evaluation

To evaluate LynX we integrated a number of different data processing operators, exhibiting different behaviours and memory access patterns and measured their performance. Additionally, we present the performance benefits of the scheduling employed by the runtime manager.We implement microbenchmarks ranging from encryption, cryptohashing and database operators to more complex real world examples like the HyperLogLog cardinality estimation algorithm, K-means operator and decision trees. We examined different scenarios with single and multiple applications running concurrently in the FPGA involving deploying as many of these operators on the FPGA as possible whilst enabling both the memory and network stacks. We compare the results to the available commercial platforms and observe the advantage of LynX due to the adaptability to different processing scenarios.

4 Conclusion

LynX is a first step to provide operating system-like features on an FPGA with the goal of making a flexible environment and ease of use whilse providing support for basic system functionality. LynX breaks with the trade-off performance/abstraction that in the FPGA world has always been resolved in favor of performance and demonstrates that generic OS-like interfaces and features are possible without performance loss.

References

[1] A. S. Dawood, S. J. Visser, and J. A. Williams. 2002. Reconfigurable FPGAS for real time image processing in space. In *2002 14th International Conference on Digital Signal Processing Proceedings. DSP 2002 (Cat. No.02TH8628)*, Vol. 2. 845–848 vol.2.

[2] Lieven Eeckhout. 2017. Is Moore's Law Slowing Down? What's Next? *IEEE Micro* 37, 4 (2017), 4–5.

[3] D. Fortún, C. G. de la Cueva, J. Grajal, M. López-Vallejo, and C. L. Barrio. 2018. Performance-oriented Implementation of Hilbert Filters on FPGAs. In *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*. 1–6.

[4] Kaan Kara et al. 2018. ColumnML: Column-Store Machine Learning with On-the-Fly Data Transformation. *PVLDB* 12, 4 (2018), 348–361.

[5] K. Kara, D. Alistarh, G. Alonso, O. Mutlu, and C. Zhang. 2017. FPGA-Accelerated Dense Linear Machine Learning: A Precision-Convergence Trade-Off. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 160–167.

[6] Kaan Kara and Gustavo Alonso. 2016. Fast and robust hashing for database operators. In *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE, 1–4.

[7] P. H. W. Leong. 2008. Recent Trends in FPGA Architectures and Applications. In *4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008)*. 137–141.

[8] X. Li, L. Ding, L. Wang, and F. Cao. 2017. FPGA accelerates deep residual learning for image recognition. In *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. 837–840.

[9] W. Lie and W. Feng-yan. 2009. Dynamic Partial Reconfiguration in FPGAs. In *2009 Third International Symposium on Intelligent Information Technology Application*, Vol. 2. 445–448.

[10] E. J. McDonald. 2008. Runtime FPGA Partial Reconfiguration. In *2008 IEEE Aerospace Conference*. 1–7.

[11] Neal Oliver, Rahul R Sharma, Stephen Chang, Bhushan Chitlur, Elkin Garcia, Joseph Grecco, Aaron Grier, Nelson Ijih, Yaping Liu, Pratik Marolia, et al. 2011. A reconfigurable computing system based on a cache-coherent fabric. In *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*. IEEE, 80–85.

[12] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 13–24.

[13] D. Sidler, G. Alonso, M. Blott, K. Karras, et al. [n.d.]. Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware. In *FCCM'15*.

[14] I. S. Uzun, A. Amira, and A. Bouridane. 2005. FPGA implementations of fast Fourier transforms for real-time signal and image processing. *IEE Proceedings - Vision, Image and Signal Processing* 152, 3 (June 2005), 283–296.

[15] Teng Wang, Chao Wang, Xuehai Zhou, and Huaping Chen. 2019. A Survey of FPGA Based Deep Learning Accelerators: Challenges and Opportunities. *CoRR* abs/1901.04988 (2019).