"Mircea cel Batran" Naval Academy Scientific Bulletin, Volume XVIII – 2015 – Issue 1
Published by "Mircea cel Batran" Naval Academy Press, Constanta, Romania // The journal is indexed in:
PROQUEST SciTech Journals, PROQUEST Engineering Journals, PROQUEST Illustrata: Technology, PROQUEST Technology Journals, PROQUEST Military Collection PROQUEST Advanced Technologies & Aerospace

LEVERAGING FPGAS AND SDNS FOR HIGH SPEED IPC IN HIGH PERFORMANCE COMPUTING CLUSTERS

Laurentiu Alexandru DUMITRU[1]
[1]Eng., Ph.D. (c), Military Technical Academy, 39-49 George Cosbuc Bvd., Bucharest, Romania, dlaur@nipne.ro

Abstract. *Many modern computing clusters have FPGA accelerators installed in their nodes. Apart from their specific purpose, these cards could also be configured to communicate with the outside world by a network interface, given that one exists. The communication structure of the cluster is, in most of the cases, a network to which nodes are connected. The migration toward the support of Software Defined Networks in switch fabric, which is already visible in several large manufacturers, gives the opportunity to dynamically create isolated networks between applications running on nodes inside the cluster. When parallel application are started on distinct physical nodes, inter process communication, with all the implications, requires special attention from system administrators and programmers. This paper explores the possibility of having an automated and transparent IPC method that is based on mapped memory, dynamically synchronized across several processes that are bound to the same SDN.*

Introduction

Current technologies have enabled both academic groups and commercial entities to take advantage of multi-core processing in condensed computing clusters to achieve high performance computing (HPC) without the need of specialized hardware and proprietary solutions. This development direction is clearly visible in public and private clouds that are built around commodity hardware. In research facilities, HPC is often achieved with open source cluster software and COTS hardware. One key element in such an environment is the communication network to which all the nodes are connected and which supports all forms of inter-node communication protocols. However, specialized clusters have dedicated accelerator cards installed in their nodes. Most of the times, these accelerators come as Field Programmable Gate Array (FPGA) cards. These cards usually connect via PCI Express and, therefore, have a substantial amount of communication bandwidth. Their main advantage is the possibility to be reconfigured to accommodate a particular application. Such installations can be seen in large clusters, as those participating on the Worldwide LHC Computing Grid at CERN, and, also in small, private ones.

Typical HPC applications often require parallelization. This happens either locally, by exploiting the multi-core/multi-thread architecture of the current host node or by distributing the processes on different physical nodes. When all the processes are contained within the same server, Inter Process Communication (IPC) is achieved with the help of the specific mechanisms of the Operating System (OS). On Unix-like systems the most used include shared Memory Mapping (mmap), Sockets and System V IPC. Memory mapping is the most efficient way to share a large amount of memory between processes. When processes are not on the same physical node, extra work is needed in order to achieve a synchronized and shared communication space. There are several approaches, among which, the most transparent and easy to implement is using a shared storage that all the nodes are connected to. This assumes that the cluster is configured in such a way that it permits this. Also by using this method, the speed is usually bound to the storage element's capabilities and, the disk wear increases. Another approach is to synchronize the data over the network, using sockets. This requires the extra programming of a multi-client – multi-server architecture and could induce other hidden catches such as firewalls or segmented networks.

Software defined networks (SDNs) are a new concept of communication networks in which the topology can be changed dynamically, according to current needs. This is achieved by decoupling the data plane from the control plane. As opposed to traditional Ethernet network which do destination-based forwarding at Layer 2, on the OSI 7 Level communication model, SDNs do rule-based forwarding. The control plane directs incoming traffic according to a preconfigured policy. Network switches that participate in SDNs must support the OpenFlow protocol. Each SDN has at least a controller to which every switch is connected. It is the controller's responsibility to configure the flow tables on each switch. OpenFlow runs on top of TCP and uses Transport Layer Security (TLS) for data privacy. Since SDNs and OpenFlow are relatively new technologies, not all vendors support them. However, several large manufacturers have started supporting these technologies. OpenFlow-capable virtual switches already exist and are deployed in cloud stacks such as OpenStack and CloudStack. Public implementations exist on reconfigurable hardware [11].

When the network of a computing cluster provides support for SDNs and nodes are equipped with FPGA accelerators, a new inter-node IPC mechanism can be implemented. This assumes that the FPGA cards have some form of external communication that is understood by the edge switch (such as Ethernet over copper or fiber optics) and have enough free space to implement the communication cores required by this method. This paper studies the concept of having virtual and on-demand private networks that connect the distributed processes, running on physical nodes inside a cluster, to them. The goal is to achieve transparent data sharing by synchronizing the memory mapped space of the processes belonging to the same execution group. The main difference from other approaches is a higher level of transparency without the performance penalty. Even if the medium is a network, the programmer does not need to think about any type of client-server communication. This is resolved by the underlying components. Since the networks are private – physical and logical, there are no issues regarding firewalls, routing or network related information. Communication is done at Layer 2 because the topology is a direct point to multipoint link where every node is directly interested in the data content. An abstract view of the system is shown in Figure 1 and is described, in detail, in chapter III.

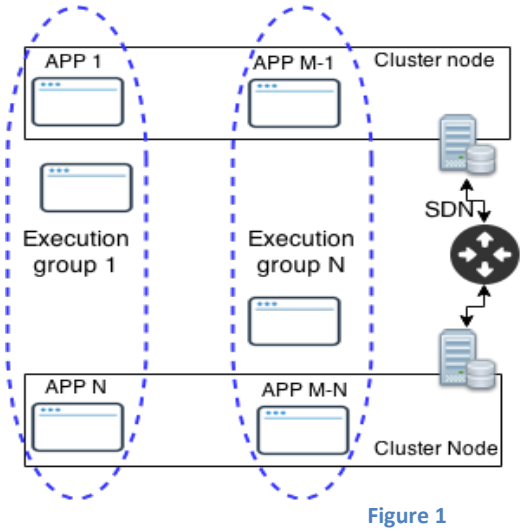

Figure 1

Related work

The proposed solution contains a blend of both emerging and production-grade concepts. The Remote Direct Memory Access (RDMA) concept is at the base of the proposed solution. In its current forms, RDMA, along with other specific HPC interconnects, is only present in clusters with dedicated

hardware (e.g. Infiniband) and software. Among the main disadvantages is the need for a distinct communication network, the implementation price and required knowledge. RDMA's role is to achieve transparent remote data transfer, without CPU intervention. It would be a good candidate to achieve intra-node IPC in a cluster, as the current paper proposes. However, the needs for specific hardware for which current software solutions are developed have lead to the development of a similar solution that is discussed in the next chapters. In particular cases [1], RDMA can be implemented on top of the existing infrastructure. Still, the need for custom software that is aware of the topology exists, which makes it less transparent and more challenging to deploy on clusters with dynamic topologies, such as clusters in a cloud environment. A similar idea can be found in [2] where RDMA is used to synchronize per-core local memories in multi-core FPGAs.

The Joint Network Interface Controller [3] project is a collaborative research project between HP and Intel to explore high-performance in-data-center communications over Ethernet. The study focuses on the real possibilities of Ethernet as an alternative to specialized Myrinet [4], Quadrics [5] and InfiniBand [6] networks to construct scalable compute clusters. In the context of 10G networks becoming a common internal interconnect in clusters and FPGAs being equipped with transceivers capable of more than 10Gbit/s it should be an advantage to combine these technologies to achieve what is normally possible only with dedicated hardware.

Since the main goal is the synchronization of shared data on distinct physical nodes, the process moving data from one node to all of the other nodes in the same communication group (CG) includes a user space - FPGA - network (and reverse) transmission procedure. Locally, moving data to and from user space and FPGA can be done in many ways. RIFFA[7] provides communication and synchronization for FPGA accelerated software using a standard interface. From a programmer's point of view, normal read/write operation can be done on a file and the date comes and goes directly from and to the FPGA. The implementation includes scatter-gather (SG) DMA with direct access to user space buffers. This assures optimal performance when moving data. RIFFA was developed as a transparent method of accessing particular IP cores that are found on the card. Opposed to RIFFA's method, the approach presented in the next chapters uses memory mapped regions, which are more suitable of synchronization in a parallel runtime environment due to the fact that the shared/manipulated data is usually malloc()'ed.

Eguro K. proposed SIRC: An extensible reconfigurable computing communication API [8] as a transparent host to FPGA communication mechanism. Like RIFFA, it uses a read/write method.

Although software defined networks (SDNs) have not yet received mainstream implementation, their flexibility is a fundamental characteristic for any dynamic cluster. The network's switching logic resides in the SDN controller [10]. This has the capability to control the full routing path of a certain packet/flow, from entry to exit. Switching is done by OpenFlow-enabled physical and virtual switches. Implmentations exist also in FPGA fabric [11]. The proposed solution does not need to be OpenFlow-aware since it connects to a SDN instead of contributing to the routing process. The SDN controller is the head of the routing decisions over the entire infrastructure. It can implement normal mac/vlan-based switching and any other type of filters like ip/tcp/udp header-based or static manual entries. It is also possible to implement dynamic (re)configuration strategies with the help of automated network observation, such as proposed in SAEN [9]. With the help of SDNs, one can achieve a transparent intra-node IPC when combined with FPGAs that have network connections.

By combining RDMA, FPGAs and SDNs it is possible to implement a new type of intra-node information sharing. This approach is documented in the following chapters.

Architecture

Figure 1 illustrates the high-level architecture. Each process is part of an Execution Group (EG), which it must declare at startup. Processes among the same EG are connected in the same virtual network which is created on demand and destroyed where there are no more members. Only one process may write to a specific region of the shared memory. Synchronization is done as in any other IPC environment. When the process that holds the protected region releases the lock, the shared region is distributed to all other processes, achieving a consistent view across the whole execution group.

As the cluster in which the FS-IPC is deployed must be SDN-capable, a SDN Controller must exist. The controller must be capable of providing an Application Programming Interface (API) through which it can be configured. This feature is needed in order to dynamically configure the networks as the processes enter and exit execution groups. Another useful API call is the one to get the (shortest) path from A to B. FloodLight[12] implements all this functionalities and runs out-of-the box, without any configuration required. Without such a function, the Software reconfiguration daemon, described below, would be responsible for computing the shortest path. The proof of concept implementation was done with this controller.

The choice was made from the comparison of five controllers [13]. When selecting the SDN controller for the test setup, the main candidates were the most used ones: POX, Ryu, Trema, FloodLight, and OpenDaylight.

The POX controller is the python-based evolution of NOX. It is mainly used for SDN debugging, network virtualization and controller design since it is very flexible.

Ryu is a component-based controller, supported by NTT. Its predefined components can be extended in order to create a custom controller. One advantage is the possibility to use any programming language for new developments.

Trema is a scripted controller which uses Ruby. It is centred around easy code and high performance.

OpenDaylight is an open source project under linux distribution. The goal of the project is to create robust code that covers most of the major components of the SDN architecture, to gain acceptance among the vendors and users, and to have a growing community that contributes to the code and uses the code for commercial products.

The FloodLight controller is java-based, making it easy to run on any operating system. It also exposes a simple but sufficient REST API.

FS-IPC is implemented as point to multipoint link, without any routing requirements. When an application commits the memory to the execution group, the contents must arrive at all nodes that are a part of the same EG. Each node in the cluster is connected to a specific port in a SDN-capable switch. This pair, switch-port, forms the source and destination identifier. Based on their participation to execution groups, the SDN must be capable of providing bidirectional communication. Being a private multicast communication group, there is actually no need for source/destination MAC. That space from the Ethernet header can be used for internal communication headers. When a node runs applications that are part of several EGs, it must be registered in the SDN topology accordingly.

The solution is composed from four different software elements that sustain the whole process.

The SDN Reconfiguration Daemon (SRD) is a server daemon that runs on the SDN controller and communicates with daemons running on cluster nodes - the Node Configuration Daemons (NCD). Its main operation can be summarized as:

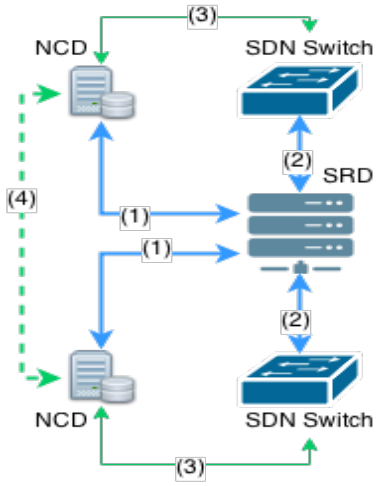

Figure 2

The node configuration daemons connect to the SRD over the standard network. Also, it connects to the SDN controller and can query information and send reconfiguration request. SRD's main task is to keep track of applications participating in EGs and to update the SDN rules in order to establish the virtual shared network environment for these.

First, the NCD signals the SRD for joining/leaving an EG(1). The SRD loops through every EG structure and retrieves the participating ports in the form of a Cartesian product from which it excludes pairs with identical elements. It then retrieves the shortest path from Source to Destination with the help of the SDN controller's API. A rule is build to reflect the path, for each pair. After computing the rules, it implements them to the required SDN switches (2). From this point, the FPGAs are connected to the same shared network (3). The result is a full mesh of virtual circuits between nodes participating in the EG (4). Each transmitted packet arrives at every node in the network, in the same group. The task of separating incoming packets according to execution groups and directing them to appropriate memory regions falls into the Node Kernel Driver's attributes. The daemon implements standard client-server architecture. Internally, execution group information is stored as:

$$EGS = \begin{cases} E1 = \begin{cases} SwitchId1 - Port1 \\ \dots \\ SwitchIdX - Port\,Y \end{cases} \\ E2 = \begin{cases} SwitchId1 - Port1 \\ \dots \\ SwitchIdX - Port\,Y \end{cases} \\ \dots . \\ En = \begin{cases} SwitchId1 - Port1 \\ \dots \\ SwitchIdX - Port\,Y \end{cases} \end{cases}$$

The Node Kernel Driver (NKD) plays a critical role, since it is responsible for transmitting and updating the shared memory regions. It must differentiate the requests for different EGs and configure the card to mark the transmissions accordingly. By having access directly to the configuration space, it must maintain consistent information about EGs that it receives. Each EG has a particular application that requested it with a specific memory region. Since the driver implements a character device, the EG number is kept in the private_data structure of the file pointer. The driver internally stores the dma regions associated with a particular EG inside a structure.

```
struct eg {
        int eg,last_buf,last_page;
        int requested_area_size;
        dma_addr_t hw_addr[NUM];
        void *buf_addr[NUM];
}
```

When mmap_open is called as a result of a userspace request, the driver has already allocated n DMA memory regions of size s, where n = requested_size / AXI-to-PCIe translation length. The zones will be mapped as DMA_TO_DEVICE when a TX signal is received and DMA_FROM_DEVICE when data is inbound. In the first case, the user application signals a transmission via an IOCTL message. In case of receiving, the application is informed, over a separate method, such as MPI, that it has data and doesn't have to do anything. The data is already copied in its memory by the FPGA logic. In this case, the driver is responsible only with setting up the appropriate registers in the FPGA card with the allocated memory regions.

When mmap_fault is called due to a memory access from userspace, the driver is responsible for mapping the dma zones to the application. Page translation is done with the standard kernel function, virt_to_page. Also, a call to get_page() is done in order to mark the page as being used. The function will cycle through every pre-allocated zone and map every page to the VM area of the calling userspace process. Boundary check is mandatory since the requested memory mapped region has a fixed size, announced to the kernel module on application initialization. On exit, the driver must free allocated resources and reconfigure the FPGA control register to reflect the leaving from an EG.

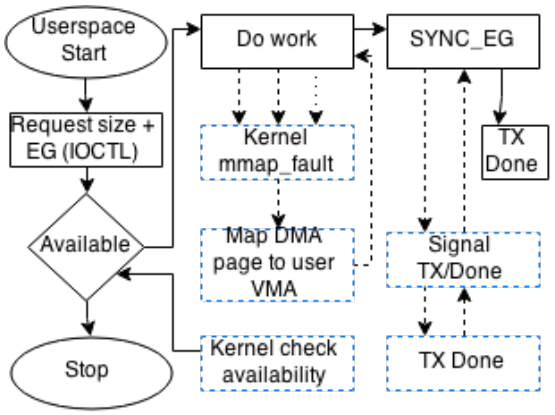

Figure 3

The Node Configuration Daemon runs locally on every node. Its main purpose is to receive information from the NKD and to signal the SRD when an application joins or leaves an execution group. An alternative to a local daemon is to link the application against a library that performs the same task. However, setting up a node-global application has several advantages, as this can be also used for monitoring the node and signaling the SRD. Furthermore, the programmer does not need to modify the code in order to call specific library functions. This way, existing applications do not need to know about the lower transport level. When an application requests joining an EG, after starting, the node kernel daemon (NKD) registers its participation on that EG by adding it as an integer element to the global EGS vector. Since the NCD is a userspace application, it must periodically query the NKD to get the required information. After the ioctl call that retrieves the list, a message is sent to the SRD over TCP/UDP that signals joining or leaving an execution group. A valid communication path between the NCD and the SRD must exist.

The FPGA logic is driven, in the prototype, by a Microblaze soft processor. Summarized, a top-level view of the architecture is presented in Fig. 4:

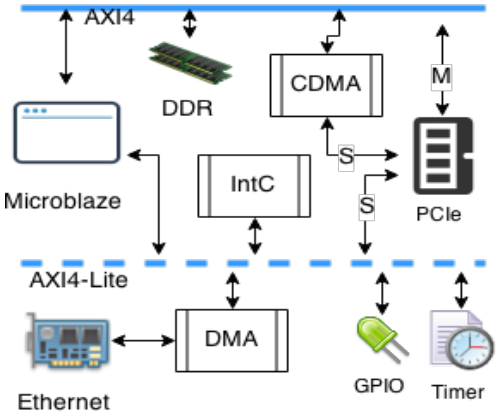

Figure 4

Implementation was done with two Xilinx boards, a Spartan 6 development kit and a Kintex kit, using the same logic. Since all used IP cores are part of the standard Xilinx library, no additional software costs exist. However, implementing a soft-core is not a mandatory requirement. This was the approach for concept validation. A production implementation should include dedicated logic for optimal speed and minimum resource usage. Apart from the softcore-related components, the design includes: two axi interconnects, one axi pcie bridge, one axi central dma controller (AXI CDMA), one axi dma (AXI DMA) controller, one axi_ethernet component, one axi_gpio and one fixed interval timer. The AXI bus connects the onboard DDR memory, the PCIe Core and the CDMA core. The CDMA is resonsible from host-initiated read/write operations destinated to the configured BAR0 address. This address space is used for configuration-related information and event signaling. The DMA controller connects to the AXI Stream interface of the Ethernet Core.

The AXI-Lite bus connects the PCIe Axi Slave interface, the Ethernet Master Scatter Gather (SG) interface and the Microblaze core. One AXI-Lite to AXI4 connector exists to facilitate access from the softcore to both AXI4 space (needed for DDR access) and AXI4-Lite space (needed for directly modifying the Host DMA memory from the softcore, during the testing phases).

Since the data transfer is done using SG-DMA, the softcore does not need to access directly the host's DMA memory. Therefore, in a production system, M_AXI_SG from the Ethernet core should be connected to the S_AXI of the PCIe core with a full AXI-4 bus, in order to benefit from transfer bursts and other performance optimizations. A different clock domain could also be used, in order to separate the existing logic from the FSIPC logic. The FIT timer is used internally for different counters. GPIO signals interrupt requests to the PCIe core, which, in turn, signals the host for a MSI interrupt request.

One important aspect of the PCIe core is that it must support dynamic configuration of AXI Base Address Translation Configuration Registers. This is needed due to the fact that the DMA zones allocated by the host are not necessarily contiguous and, the core can access only a specific memory rage starting from a base address. By supporting dynamic translation, the core's starting PCIe address is reconfigured when read/write is performed on a new DMA zone. These registeres are configured by the NKD at load time. Also, BAR0 length must be the same size as required for DMA transfers. When translated from PCIe to AXI, the BAR0 memory region serves as a signaling and register zone. It is divided as follows:

Run Flag	Current EG	Num DMA	DMA Hw_addr1	..	DMA addrN

The Run flag is used for TX signaling. When an application issues a SYNC command, the NKD fills out the registers with their appropriate values, sets TXSTART as the run flag and waits for its change to TXDONE. The change is announced by an interrupt from hardware. After the transmission is complete, the flag is reset to IDLE. There is no need for an RX mechanism since the receive side is done from the card and, at the time when the application receives the signal that it can access the data, the DMA zone are already filled with it. As the Ethernet core starts to receive packets, it extracts the execution group from the packet's header. It then sets the Runflag to RXINIT and issues an interrupt. The NKD reads the Current EG from BAR0, fills out the DNS zones and sets the flag to RXSTART. As the pages are received, they are written to the appropriate DMA zones. Given this execution delay, if the Ethernet core does not have enough buffer space to hold the incoming frames, these can be saved in the local DDR memory and transferred at the end to the host memory. One alternative is that the sending core to begin with a configuration frame that has no content but contains the execution group. When received, pages are written in the incoming order. The system could be extended to support synchronization of specific memory areas inside the shared space. This would imply adding extra fields in the packet headers that indicate a starting offset inside the memory space and a number of pages to be written.

Test results and further optimizations

The tests were carried out in a controlled environment consisting of two nodes with FPGA cards, one node that is also a part of the SDN execution group and one controlling node. The Open vSwitch is hosted on a system with multiple network cards that are jumbo frame capable. Both nodes were based on Intel Core 2 Duo Processors, 4GB of RAM and CentOs Linux 7 running with 3.10 kernel. Speed measuring on the network was done with libpcap. Wireshark was used for traffic analysis on the monitoring node. In-kernel timing was done with the do_gettimeofday API call. Overall, the transfers to and from the FPGA worked as excepted and the tests have validates the proposed FPGA-SDN interprocess communication solution. Although specialized solutions perform better in certain cases, the proposed no-cost alternative for clusters that already have the required hardware can be, at least, a concept validation solution.

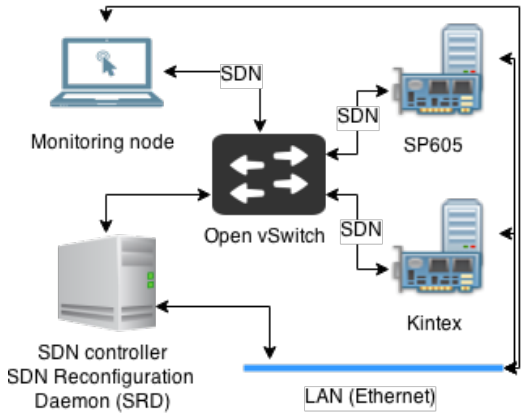

Figure 5

The majority of the network switches are jumbo frame capable, with sizes up to 9KB per frame. Given that, on most of the server operating systems, the page size is 4KB. One packet can contain the control header and two pages of memory. This approach lowers the interrupt frequency, reducing CPU usage, but, increases the RX buffer size due to the large frame length. The AXI bus was configured with 32bit data width. FIFOs on transmit for MemoryMap2Slave interface and/or receive for Slave2MemoryMap interface were not used due to the fact that routing did not succeed on the Spartan board. The maximum allowed frequency for the SP605 board is 62.5MHz, limitation given by the PCIe core. The test board was actually ran at 50MHz, giving a maximum theoretical bandwidth of 1.6 Gbit/s instead of 2Gbit/s for PCIe 1.0 (2.5GT/s). It is enough to saturate the network link.

After the test setup has validated the correct functionality of the system, further optimization – in both speed and area terms – is the next step. According to official vendor documentation, resource utilization for the components is:

		LUT	Slice	BRAM
DMA	Kintex	3125	1418	2
	SP605	2934	1128	4
Ethernet	Kintex	4770	2247	26
	SP605	4877	2398	44

Depending of the selected setup, MicroBlaze can occupy from 1000 to more than 3000 LUTs. Since the majority of the accelerator cards use specially designed cores, it is highly unlikely that there will be a softcore present on a FPGA card installed on a server. In the test setup, the softcore was used solely as a control mechanism. It should be replaced with the appropriate FSM that will free up resources and speed up the system. As discussed in Chapter III, the Ethernet's DMA SG engine was connected using AXI4 instead of AXI4Lite to the PCIe Slave Bridge, for full performance. On the host system, each DMA zone was 4MB long. The test was carried out on a 128MB zone, summing up 32 smaller zones. A larger zone was used due to the fact that the PCIe core must be dynamically reconfigured to access a certain memory region from the host system. If, for example, 128Kb zones were used, then 1024 reconfigurations were needed to access the whole memory – as opposed to only 32 for 4MB zone. After each reconfiguration, the buffers need to be reconfigured and transmission restarted on the Ethernet core. Since the clock is only 50MHz, the test system used the maximum DMA zone size on Linux, which is PAGE_SIZE* 2^10 = 4MB, to achieve higher throughput.
Each buffer descriptor on the FPGA core is 64KB. The SG engine allows frame composition from multiple buffer descriptors. Tests were done using 256 descriptors that were dynamically allocated and freed as long as there was data to transmit. Since the host kernel driver only maps pages, the header is generated on the FPGA. Each frame is composed of two buffer descriptors: one that signals the SOF (Start of Frame) and points to the header, inside the FPGA memory range and another one that signals EOF (End of Frame) and points to the actual data on the PCIe memory space. This method allows direct copy to/from the DMA zone, without the need of an intermediary copy for frame composition.
From the tests summarized in Fig. 6, the best results were, as expected, with the largest payload. As the payload size decreases, more packets are needed to complete the transfer. Being in SG mode, each transmission round requires clearing and reconfiguring the buffer descriptors. This translates in CPU cycles used by the Microblaze and subsequent components. Therefore, the throughput decreases with the payload size.

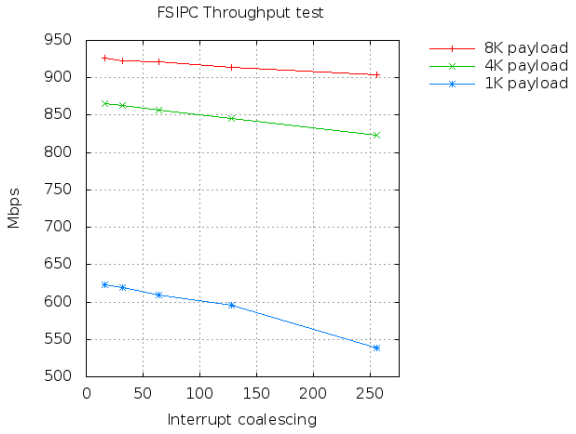

Figure 6

The proposed architecture for the FPGA cores will be different, according to the hardware used and implementations needs. Nevertheless, optimization strategies should be considered, regardless of the device.
In terms of speed, the system can be redesigned to include a separate clock domain for the PCIe core that restricts the AXI clock to 62.5MHz for the SP605 board. Instead of a memory mapped PCIe bridge, an AXI4-Stream based implementation can be used. The Ethernet's DMA engine would no longer be needed. Instead a multiplexer for the incoming streams that will ensure connectivity between the custom accelerator cores and the Ethernet core will have to be instantiated. Also, with this approach, the header building stage will reside into the kernel code.

Conclusion

The proposed solution for intra-node data sharing in a cluster is feasible as a concept, given that the cluster is SDN ready and at least some of the computing nodes have FPGA accelerators. Leveraging existing hardware in order to optimize the computing architecture inside a cluster which mostly runs applications that require parallelization should be taken into account by both cluster administrators and application programmers. Since the solution only extends transparently from node-local to intra-node, the fundamental problems and solutions for parallel applications are the same. From the runtime point of view, it is safe to compare N nodes that participate in the same execution group with one big node with M cores. Inherently the applications must implement the same synchronization, run order and access control procedures as they would in a single server with multiple computing cores. The obvious advantage is the transparent scaling capability.
If technologies will continue to develop towards the abstractization of the underlying hardware, it is possible to envision a runtime environment in which normal applications will use memory, file descriptors and even system calls across multiple nodes (physical or virtual), without particular programming requirements. The solution proposed in this paper takes a step into such a scenario, a scenario in which new types of computing clouds could emerge.

Bibliography

[1] Oberg, Michael, et al. "Evaluation of rdma over ethernet technology for building cost effective linux clusters." 7th LCI International Conference on Linux Clusters: The HPC Revolution. 2006.
[2] Kachris, Christoforos, et al. "Network processing in multi-core FPGAs with integrated cache-network interface." Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on. IEEE, 2010.
[3] Schlansker, Michael, et al. "High-performance ethernet-based communications for future multi-core processors. Proceedings of the 2007 ACM/IEEE conference on Supercomputing. ACM, 2007.
[4] N. J. Boden, et. Al.. "Myrinet: A gigabit-per-second localarea network." IEEE Micro, vol. 15, no. 1, pp. 29-36.
[5] F. Petrini, et. al. "The Quadrics Network: High Performance Clustering Technology", IEEE Micro, Feb. 2002, pp. 46-57.
[6] J. Liu, et. al. "High Performance RDMA-Based MPI Implementation over Infiniband", Proceedings of the 17th Annual Conference on Supercomputing, June 2004, pp. 295-304.
[7] Jacobsen, Matthew, and Ryan Kastner. "RIFFA 2.0: A reusable integration framework for FPGA accelerators." Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on. IEEE, 2013

[8] Eguro, Ken. "SIRC: An extensible reconfigurable computing communication API." Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on. IEEE, 2010.
[9] Maniu, Rares, L.A. Dumitru. "Self-adaptive networks with history extrapolation, evolutionary selection and realtime response." Optimization of Electrical and Electronic Equipment (OPTIM), 2014 International Conference on. IEEE, 2014.
[10] Shah, Syed Abdullah, et al. "An architectural evaluation of SDN controllers." Communications (ICC), 2013 IEEE International Conference on. IEEE, 2013.
[11] Naous, Jad, et al. "Implementing an OpenFlow switch on the NetFPGA platform." Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. ACM, 2008.
[12] Fernandez, Marcial P. "Comparing openflow controller paradigms scalability: Reactive and proactive." Advanced Information [13] [13] Networking and Applications (AINA), 2013 IEEE 27th International Conference on. IEEE, 2013.
Khondoker, Rahamatullah, et al. "Feature-based comparison and selection of Software Defined Networking (SDN) controllers." Computer Applications and Information Systems (WCCAIS), 2014 World Congress on. IEEE, 2014.