A CAM-Free Exascalable HPC Router for Low-Energy Communications

Caroline Concatto(✉), Jose A. Pascual, Javier Navaridas, Joshua Lant, Andrew Attwood, Mikel Lujan, and John Goodacre

School of Computer Science, University of Manchester, Manchester, UK
caroline.concatto@manchester.ac.uk

Abstract. Power consumption is the main hurdle in the race for designing Exascale-capable computing systems which would require deploying millions of computing elements. While this problem is being addressed by designing increasingly more power-efficient processing subsystems, little effort has been put on reducing the power consumption of the interconnection network. This is precisely the objective of this work, in which we study the benefits, in terms of both area and power, of avoiding costly and power-hungry CAM-based routing tables deep-rooted in all current networking technologies. We present our custom-made, FPGA-based router based on a simple, arithmetic routing engine which is shown to be much more power- and area-efficient than even a relatively small 2K-entry routing table which requires as much area and one order of magnitude more power than our router.

1 Introduction

Exascale computing is the next challenge for the supercomputing community aiming to design systems capable of delivering Exaflops (10^{18} floating point operations per second). To achieve these huge computing capabilities, systems will require millions of interconnected computing elements to execute massive parallel applications. Traditionally these were High Performance Computing (HPC) applications where the computation:communication ratio was heavily biased towards the former. However, the wider availability of increasingly large computing facilities and the new paradigms associated to the ubiquitous digital economy have favored the emergence of new data-oriented applications arising from the massive amounts of scientific- or business-oriented data that are being generated. These new application domains (e.g. MapReduce [9], graph-analytics [7] or new HPC database systems such as MonetDB [21]) impose completely different needs to the computing systems (and specially to the interconnection and I/O subsystems). In order to suit the necessities of these new kind of data-intensive applications, new architectures and platforms are being developed, such as our

J. Goodacre—This work was funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671553.

M. Berekovic et al. (Eds.): ARCS 2018, LNCS 10793, pp. 99–111, 2018.
https://doi.org/10.1007/978-3-319-77610-1_8

novel, custom-made architecture, ExaNeSt [14]. In such systems the Interconnection Network (IN) is crucial to ensure performance, mainly because it needs to support extreme levels of parallelism with applications using tens of thousands of nodes with any latency or bandwidth bottlenecks translating into severe penalties to execution time.

One of the main limitations for the scalability of HPC (and datacentre) facilities is power consumption. The largest current systems based on traditional HPC processors are over one order of magnitude[1] away from Exascale but already require a large, dedicated power station to supply electricity to the system. If we tried to scale computing systems just by putting more components together without changing the architectures or paradigms, we will end up requiring tens of power stations, just to power the system, which is obviously unattainable. Some steps towards reducing power have been taken in the computing subsystems by using ARM processors [4] or accelerators (e.g. GPGPU or FPGAs) that offer high FLOPs/Watt ratios. However, improving the efficiency of other subsystems has been typically ignored. For instance, the network can account for over 10% of the total power during peak utilization and up to 50% when the system is idle [1]. Other authors mention more conservative, but still significant power breakdowns in the range 10–20% [13]. This large share of the power bill of such systems motivates our search for more power-efficient IN designs.

In this regards, we notice that most networking technologies, e.g., Infiniband or 10 Gbps/100 Gbps Ethernet, rely on huge routing tables which are typically implemented as content addressable memories (CAMs). CAMs are an integral part of the design and, indeed, tend to be much bigger than the router logic itself (i.e. buffers, crossbar, flow control mechanisms, etc.). This is because tens of thousands of entries need to be stored to be able to reach all the nodes of the system [22]. In addition, routing tables create other scalability issues. First, as the size of the system increases, the size of the tables (number of entries) needs to grow accordingly. Furthermore, given that routing tables have information distributed across the whole system, they are quite sensitive to routing inconsistencies and, obviously, consistency mechanisms are in themselves another limit to scalability. All the reasons above motivate our design where we get rid of routing tables to achieve substantial savings in terms of area and power footprint. Our FPGA-based router relies on simple arithmetic routing instead. For the purpose of this work we have considered common topologies (fattree [18], dragonfly [16]) but other topologies are possible. Our experiments measure area and power consumption for varying number of ports and CAM entries. Results show that routing tables are not only prohibitive in terms of area, since a relatively small CAM uses more area than a 16-port router, but also that they can consume the whole power allowance of the FPGA.

[1] See www.top500.org.

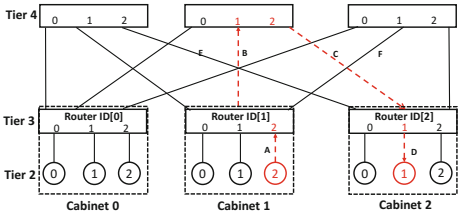
2 Related Work

One of the first steps towards using FPGA for networking was the NetFPGA [25] project which provides software and hardware infrastructure for rapid prototyping of open-source high-speed networking platforms. NetFPGA platform enables to modify parts of it and compare with other implementations. However, there are many differences between NetFPGA and our home-made router. First of all, NetFPGA focuses on IP networks and, thus, relies on routing tables, which as explained we want to avoid. Moreover, IP networking has many overheads that dismiss it as a good infrastructure for HPC networks due to inadequate throughput and latency. Finally, the NetFPGA platform has many features that consume lots of area and power but are not required in the context of ExaNeSt.

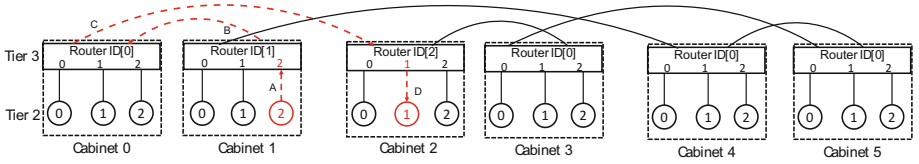
While arithmetic routing *per se* is not a new idea, its use in recent years has been restricted to cube-like topologies such as the ones in the BlueGene family of supercomputers [6] or the TOFU interconnect [2]. To our knowledge, flexible architectures relying on arithmetic routing, but capable of being arranged into different topologies just by reconfiguring the firmware (to update the routing logic) such as the one we introduce here have never been proposed before. Arithmetic routing is commonly used in SW to fill the routing tables of the switches of table-based technologies (see, e.g., [23] which generates routes arithmetically and then embed them in the routing tables of an Infiniband IN). There also exist more advanced strategies (also for Infiniband) that take into consideration the congestion of the links by storing this information in the routing tables together with the destination address to perform routing decisions [24]. More recently, the Bull EXascale Interconnect (BXI) [10] has followed a similar approach. They use a 2-stage routing strategy [22]: first an off-line algorithm calculates the paths between each source and destination. These paths are deterministic and populated into the routing tables during system start-up (could be done arithmetically). The second stage is performed on-line, when the system is running, and can change the previously calculated static routes in order to avoid congestion or failures. The 48-port routers, implemented as ASICs, store 64K entries for each port for a total of 3M entries per router. Bull switches use 2 routing tables, a bigger one with the addresses set at start-up and another small table used in case of faults or congestion in which the addresses are used to repair faulty routes.

The only effort on minimizing the impact of routing tables on the networking equipment we are aware of is on strategies to reduce their footprint. For example, using a 2-level CAM routing strategy [3]: the first level stores addresses that require a full match in order to select the output port, the second level stores masks. If the first level does not produce a match, then the selection of the port is performed based on similarity between the mask on level 2 and the destination address. This helps alleviating the impact of routing tables in terms of area and power to some extent, but the other scalability issues of routing tables still hold.

Alternatives to local CAMs do exist, but none of them would keep appropriate performance levels for FPGA-based HPC interconnects. For instance, using an off-chip CAM would severely slow packet processing because of the extra



(a) ExaNeSt arranged as a Fattree topology



(b) ExaNeSt arranged as a Dragonfly topology

Fig. 1. ExaNeSt system-level networks with route examples in red (2, 1 to 1, 2). (Color figure online)

delays to go off-chip for routing information. Moreover in a extreme-density design, such as the one we propose in ExaNeSt, adding extra components to the already tightly packed boards is undesirable. Implementing the tables in RAM (as some low-end switches do), would render information fetching even slower due to the lack of parallel access. A proposal that assigns range(s) of addresses to ports [12] and routes to the port which matches the destination was a step towards getting rid of CAMS. However, it is restricted to tree-like topologies and does not scale very well for large networks because range complexity increases with network size.

3 ExaNeSt System Architecture

In this Section we introduce the architecture of ExaNeSt, which will be showcased by means of a small, 2-cabinet, prototype—currently under construction. An ExaNeSt system will require millions of low-power-consumption ARM+FPGA MPSoCs to reach Exascale and includes a unified, low-latency IN and a fully distributed storage subsystem with data spread across the nodes using local Non-Volatile Memory (NVM) storage. Our building block is a quad-FPGA-daughterboard (QFDB) based on Zynq Ultrascale+ MPSoCs[2]. The next level (Tier 1) is the *Blade*, which is composed by up to 16 QFDBs interconnected using a backplane that delivers high-bandwidth connectivity, whilst reducing the costs and power consumption of external cables and transceivers. Six of these Blades are contained in a *Chassis* which also incorporates our FPGA router with a variable number of links that are used to interconnect the blades (Tier 2) as well

[2] See https://www.xilinx.com/support/documentation/white_papers/wp482-zu-pwr-perf.pdf.

as to provide uplinks to the system-level interconnect (Tier 3 and above depicted in Fig. 1). As these routers are implemented on FPGAs, the number of uplinks can vary in order to deliver networks with different characteristics. Next, we will focus on describing the architecture of the FPGA-based router used in Tier 3 (and above).

3.1 Router Architecture

The architecture of the router (inside the red square) is depicted in Fig. 2 together with the FIFOs, MACs and PHYs. We built a 3-stage pipelined router using a wormhole switching approach in which the packets (composed of header, payload and footer) are split into multiple flits of size 66 bits (64 bits for data and 2 extra bits to control the beginning and the end of the packets). The router sends and receives flits from and to the FIFO using a handshake flow control mechanism implemented using two signals: `val` and `ack`. When data is ready to be sent in the FIFO the `val` signal is enabled; if there is space to store the data into the router, the `ack` signal will be enabled. When at some point there is no more data available in the FIFO or no more space at the router, the corresponding signal will be disabled. A similar process happens in the output ports. The data sent and received by the FIFO comes from and goes to the 10 Gbps custom-made MAC layer which is connected to the 10 Gbps transceivers (PHY), which serialize/deserialize the data between the routers using an optical fiber. Our router uses *Virtual Output Queues* (VOQs) [8] to reduce Head of Line (HOL) blocking and, in turn, minimize congestion. Although the use of VOQs increases resource utilization, we expect the extra resources to be compensated by the performance gains and the savings of our table-free design.

The three stages of our router are as follows. **Stage-1:**, the router receives the `val` signal (a new packet has arrived to an input port). The header flit will be stored in a register. **Stage-2:** the *arithmetic routing block* decides, based on the destination address of the packet, the output port to forward the packet. Then the desired VOQ is selected and used for the remaining flits of the packet. **Stage-3:** the switch allocator selects one input port (among all the requesting ones) to be forwarded through the crossbar to the required output port. For simplicity we use round robin arbitration, but others are possible.

3.2 Routing Algorithms

Our protocol relies on a geographic addressing scheme in which the location of all the components is embedded in their address. This comes as a side-effect of the highly hierarchical system. The current prototypes would require 22 bits out of the 24 available for encoding end-point ids (2 bits for the chip within a DB, 4 bits for the DB within a mezzanine, 4 bits for the mezzanine within a chassis, 4 bits for the chassis within a cabinet and 8 bits for the cabinet). This would leave 2 free bits within an address that could be used for different purposes, e.g., multipath routing, priority levels or system-level operations. Such a naming convention is enabled by the fact that FPGAs come without a defined address and that

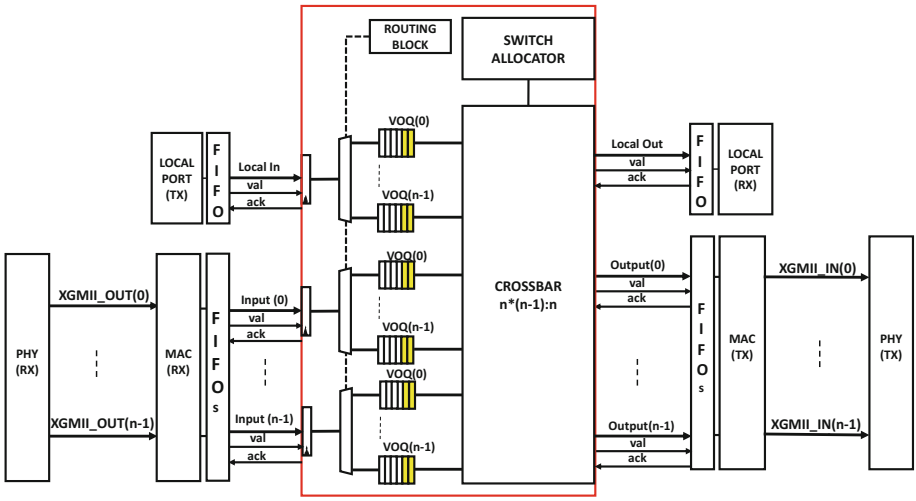


Fig. 2. Block diagram of the 3-stage router plus the FIFOs, the MACs and the transceivers (PHYs). (Color figure online)

initializing it at boot-up time would be trivial and would require barely any overhead, just by leveraging locational information into the different levels, e.g. through system-level controllers or even an EPROM holding this information.

It is our vision that having this hierarchical information within the addressing scheme can be exploited by means of arithmetic routing as many high performance topologies feature very simple routing algorithms that take routing decisions based only on a single coordinate within the hierarchy (e.g. k-ary n-trees and Dragonfly, as provided here or others such as generalised hypercubes [5], Clos [20] or torus [2,6]). Indeed, such arithmetic forms of routing are specially well suited for FPGAs as they would require very simple logic to be implemented and could be changed accordingly to the selected topology as opposed to an ASIC-based implementation, which must be static (or software based).

Algorithm 1 shows the routing algorithm for a fattree. Packets travel up and then down the tree according to the destination address, tier and router ID. This is done in order to avoid deadlocks [19]. First, the algorithm checks if the router is in Tier-4 (the top of the tree), in which case the packet goes down through the port connected to the destination cabinet. If the packet is in Tier-3, the router checks if the destination address is local to its cabinet, in which case it takes the port connected to the corresponding chassis. Otherwise the packet goes through any of the uplink ports (using Round Robin for simplicity), seamlessly performing multipath routing. In the future we expect to investigate improved congestion-aware policies. Figure 1a, shows a route example marked with red dotted lines. We denote addresses as $[Cabinet, Chassis]$. The source, $[1, 2]$, sends a packet to the destination $[2, 1]$. First $[1, 2]$ sends the packet to router 1 in Tier 3 using link A. Then, the packet will be sent through any uplink (B, in the

Algorithm 1. Routing strategy for fattree

```
1: procedure RouteFattree(header, tier, routerId)
2:     if tier = 4 then                                          ▷ top tier
3:         req ← header.cabinet
4:     else if header.cabinet = routerId then
5:         req ← header.chassis                                  ▷ go down
6:     else
7:         req ← port going up                                   ▷ go up
8:     return req
```

Algorithm 2. Routing strategy for dragonfly

```
1: procedure RouteDragonfly(header, routerId)
2:     if header.cabinet = routerId then
3:         req ← header.chassis                                  ▷ go down
4:     else if group(header.cabinet) = group(routerId) then
5:         req ← intraGroupPort(header.cabinet)                  ▷ same group
6:     else
7:         req ← interGroupPort(header.cabinet)                  ▷ route to other group
8:     return req
```

example) to Tier 4, because of line 7 in Algorithm 1. Now the packet is in Tier 4, so Algorithm 1 dictates to follow link C to Cabinet 2 (line 3) and the packet arrives to router 2 in Tier 3. Now the router ID and destination Cabinet are the same, so line 5 in Algorithm 1 selects port 1 (Chassis of destination address is 1) and the packet is forwarded through link D. Finally, the packet arrives to $[2, 1]$, and is routed to the correct QFDB through the lower Tier networks.

Algorithm 2 shows the routing algorithm for dragonfly. Packets travel between groups according to the destination address and router ID. First the algorithm checks if the packet is addressed to the local router, in which case the packet goes down to the corresponding chassis. If not the router checks whether it goes to another cabinet in the group in which case it takes the port connected to the corresponding router. Otherwise the packet needs to move to a different group, either directly through their up-ports or through another router in the group through the intra-group ports. Functions `group()`, `intraGroupPort()` and `interGroupPort()` are arithmetic and use router coordinates and topology parameters only, but are not shown here due to space constraints. Figure 1b shows a route example between nodes $[1, 2]$ and $[2, 1]$. First $[1, 2]$ sends the packet to router 1 in Tier 3 using link A. Then, the packet will be sent to router 0 through link B, as dictated by line 7. Given inter-group routing is still needed Router 0 will forward to Router 2 following link C, (line 7). Now the router ID and destination Cabinet are the same, so line 3 selects port 1 and the packet is forwarded to the destination chassis through link D.

4 Evaluation

In this Section we firstly present our set-up to measure the area, power and performance (Throughput and latency) required to implement the router and the routing tables. FPGAs have a restricted amount of resources and router design must scale nicely, i.e., do not explode in terms of resources (or power) as the number of ports or the size of the CAMs increase. Therefore we measure the area and power consumption of the approaches to show their scalability. Finally we measure throughput and latency as they are the most important performance metrics for HPC systems.

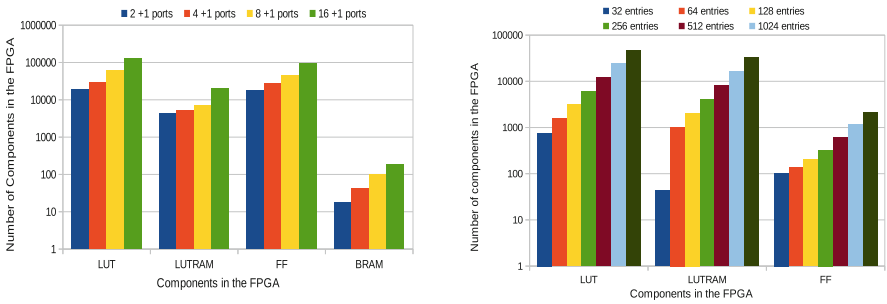
4.1 Experimental Setup

We implemented the router architecture described in Sect. 3 (and shown in Fig. 2) as a soft core IP in Verilog and synthesise it in a Virtex-709 FPGA. The transceivers (PHY in Figure 2) are hard-core IPs in the FPGA containing a serializer/deserializer (serdes) IP working at 10 Gbps and 156.25 MHz[3]. We use a custom MAC IP which synchronizes the clocks between the transceivers of the sender and the receiver by adding a short preamble and footer in the packets. Finally the router was instantiated with a varying number of ports plus one local port (used as injector/consumer for testing purposes). The FPGA area is measured and considers the amount of Look-up-Tables (LUT), LUTRAM (LUT used as memory), Flip-flops (FF) and Memories (BRAM) consumed by the router and the routing table. To measure the performance, we used two interconnected Virtex FPGAs. In this experiment, the router has 4 external ports plus one local port because our development boards have only 4 SFP ports. Thus the routers were instantiated with 3 downlinks + 1 uplink. The two boards were wire connected using optic fibers and the traffic was generated and received by soft-core MicroBlaze processors attached to the local ports. Traffic was composed of packets with 100 flits length generated at intervals of 11 clock cycles. We provide the local port interface with counters to measure the number of packets received in 1 s and the delay to receive the first packet after the system has started.

4.2 Area

The BRAMs were used to implement the buffers in the MAC layer. The LUT and FFs were used to implement the logic and the LUTRAMs were used to implement the VOQs (with space for 16 flits each) and the memory in the routing table. In case of the routing tables, for the sake of clarity, we just show the resources used to implement them, not the whole router. Given that the footprint of the arithmetic routing block is negligible, implementing the switch with the routing tables will require, at least, the same amount of resources as implementing each of them separately.

[3] See seen on 8th January 2018: https://www.xilinx.com/support/documentation/ip_documentation/ten_gig_eth_pcs_pma/v6_0/pg068-ten-gig-eth-pcs-pma.pdf.



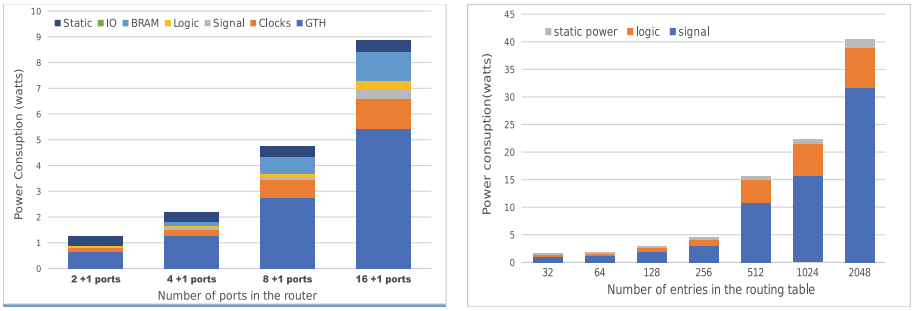
(a) Arithmetic router (b) Routing tables

Fig. 3. Area used in the FPGA.

Figure 3a shows the area results for our arithmetic router, including the MACs and PHYs, the latter two takes most of the router area. We have measured the area of between 2 and 16 ports (plus the local port, used for evaluation purposes). The maximum area required in the FPGA is around 30% of the LUTs for the 16-port version. More importantly, resource consumption scales roughly linearly with the number of ports which show the scalability of our design. For comparison, Fig. 3b shows the area required to implement routing tables with different number of entries, from 32 up to 2048. Routing tables were implemented following the node-table approach shown in [11] in which one table is shared among all the input ports of the router. The logic of the routing table will match the destination address with the stored node addresses and then, extract from that CAM line the output port to be used. The area required increases roughly linearly with the number of entries requiring almost 20% of the LUTRAMs for 2K entries. Even for a relatively small routing table by today's standards (e.g. 64K entries used by Bull interconnect [10], 48K for Infiniband or 32K for Ethernet [15]) these routing tables take a significant part of the FPGA resources and would seriously limit the scalability and the number of ports we could implement. Moreover tables with 256 entries or more cannot work at our target frequency, as shown in Fig. 3b. This is because the huge MUX/DEMUX trees required to access the tables severely increase the critical path. Comparing the routing table area with the router is not trivial as the routing tables uses more LUTRAM to implement memory and the router uses more FF and LUTs to implement its logic. In terms of LUTs a 4-port router consumes almost the same as a routing table with 1K entries. However for LUTRAMs a 4-port router uses almost the same area as a 256-entry routing table.

4.3 Power Consumption

Figure 4a shows the power consumption estimated by Xilinx tools for routers with 2, 4, 8 and 16 ports (plus the local port) for the different resources used by the router. Notice that the GTH transceivers work at a frequency of



(a) Arithmetic router (b) Routing tables

Fig. 4. Power consumption in Watts.

156.25 MHz in order to transmit at 10 Gbps. However the router with 16 ports works at a slightly lower frequency, 140 MHz, due to the size of the crossbar that grows exponentially. Other aspect of the implementation in the FPGA that should be noticed is that Virtex-709 only has 4 SFP+ connectors (hence 4 GTH transceivers). For that reason the remaining serial ports for the 8- and 16-port routers were placed in the FMC HPC connector (standard connection for any type of interface) of the FPGA using the same clock source (SFP+ connector). The results clearly show that the GTH transceivers are the resources that consume more power (higher than 50% of the total). The maximum power consumption (for 16 ports) is ∼4 W which is relatively low; about 10% of the max FPGA power (40 W) for this implementation. Figure 4b shows the power consumed by the routing tables measured at a frequency of 156.25 MHz. We show both the dynamic (logic plus signal switching) and the static power. As expected the routing tables demand higher power as we increase the number of entries for the same frequency (156.25 MHz). For instance, a routing table with 2048 entries consumes 40 W, which is already the maximum FPGA power rendering the implementation of other elements impossible. In contrast 32 entries consumes less then ∼4 W. A 8-ports router plus MACs and PHYs consumes the same power as 256 entries routing table. Moreover a router with a routing table with 2K entries consumes almost 4 times more power than the proposed routers with 16 ports + 1.

4.4 Performance

We close this Section by measuring the throughput and latency of our design. Notice that a VOQ-based router using routing tables would theoretically have the same performance as ours (assuming that accessing the table can be done in one clock cycle and that the frequency remains the same). Our tests showed that the router is able to maintain a throughput higher than 8 Gbps (with 10 Gbps transceivers), which is acceptable for a first prototype. The main culprit for not being able to saturate the links (achieve 10 Gbps) is our custom

MAC implementation which stalls packet-forwarding in order to check whether the transceivers are synchronized. Regarding the latency per hop, our measurements drew between 70 to 80 clock cycles to traverse both routers. This latency is the time required to traverse the source router (3 cycles), MAC (12 cycles), both transceivers TX and RX (25–45 clock cycles each) and the MAC in the destination router (12 cycles). Note that data transmission is much slower than taking routing decisions in our design.

5 Conclusions and Future Work

The interconnection network will play a crucial role in future systems that aim to break the Exascale frontier. One of the main concerns in these systems is the reduction of the power consumption, issue that is being faced by using low-power computing elements or other power-efficient devices delivering high performance/Watt. However in these massive interconnected systems the network can be responsible of consuming a large share of the required power, so traditional approaches are not suitable any more. To deal with this issue we propose a disrupting interconnection architecture that avoids the use of costly and power hungry routing tables. These are deep-rooted in commercial devices for HPC and datacentre networks. Our design leverages an FPGA-based arithmetic router with our geographical addressing scheme.

Our experimental work shows that the amount of resources required to implement the router is small allowing designs with more than 32-ports in this particular FPGA model. Regarding the power consumption the routing tables exceed the maximum power output of the FPGA as early as 2K entries. On the other hand, the router implemented using the arithmetic routing requires less than 5 W, that is, 12.5% of the power delivered by the FPGA. Finally we measured the throughput and latency showing promising figures of 8 Gbps and 70–80 cycles (500 ns) per hop, respectively. Moreover, we found that avoiding the use of routing tables is essential for our design as a small CAM table (2K entries), would not only require ∼20% of the FPGA resources, but would also exhaust the power budget of the FPGA. In the future, we plan to improve the performance of the router optimizing the MAC layer. We will also evaluate the area and power consumption of the arithmetic router using more modern FPGAs like the Virtex UltraScale+ from Xilinx. Finally we want to explore the impact of our VOQs + arithmetic router on the performance of larger networks by using our in-house developed simulator, INSEE [17].

References

1. Abts, D., et al.: Energy proportional datacenter networks. In: International Symposium on Computer Architecture, ISCA 2010, pp. 338–347. ACM, New York (2010)
2. Ajima, Y., et al.: The Tofu interconnect. IEEE Micro **32**(1), 21–31 (2012)

3. Al-Fares, et al.: A scalable, commodity data center network architecture. In: ACM SIGCOMM 2008 Conference on Data Communication, SIGCOMM 2008, pp. 63–74. ACM, New York (2008)
4. Aroca, R.V., Gonçalves, L.M.G.: Towards green data centers: a comparison of ×86 and ARM architectures power efficiency. J. Parallel Distrib. Comput. **72**(12), 1770–1780 (2012)
5. Bhuyan, L.N., Agrawal, D.P.: Generalized hypercube and hyperbus structures for a computer network. IEEE Trans. Comput. **33**(4), 323–333 (1984)
6. Chen, D., et al.: Looking under the hood of the IBM Blue Gene/Q network. In: Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–12, November 2012
7. Cuzzocrea, et al.: Big graph analytics: the state of the art and future research agenda. In: Proceedings of the 17th International Workshop on Data Warehousing and OLAP, DOLAP 2014, pp. 99–101, ACM, New York (2014)
8. Dally, W., Towles, B.: Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers Inc., San Francisco (2003)
9. Dean, J., et al.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
10. Derradji, S., et al.: The BXI interconnect architecture. In: IEEE Annual Symposium on High-Performance Interconnects, HOTI 2015, pp. 18–25. IEEE Computer Society, Washington (2015)
11. Duato, J., et al.: Interconnection Networks: An Engineering Approach. Morgan Kaufmann Publishers Inc., San Francisco (2002)
12. Gómez, C., et al.: Deterministic versus adaptive routing in fat-trees. In: Workshop on Communication Architecture on Clusters (CAC 2007) (2007)
13. Heller, B., et al.: ElasticTree: saving energy in data center networks
14. Katevenis, M., et al.: The exanest project: interconnects, storage, and packaging for exascale systems. In: 2016 Euromicro Conference on Digital System Design (DSD), pp. 60–67, August 2016
15. Kieu, T.C., et al.: An interconnection network exploiting trade-off between routing table size and path length. In: International Symposium on Computing and Networking (CANDAR), pp. 666–670, November 2016
16. Kim, J., et al.: Technology-driven, highly-scalable dragonfly topology. In: 2008 International Symposium on Computer Architecture, pp. 77–88, June 2008
17. Navaridas, J., Miguel-Alonso, J., Pascual, J.A., Ridruejo, F.J.: Simulating and evaluating interconnection networks with insee. Simul. Model. Pract. Theory **19**(1), 494–515 (2011). http://www.sciencedirect.com/science/article/pii/S1569190X1000184X
18. Petrini, F., Vanneschi, M.: k-ary n-trees: high performance networks for massively parallel architectures. In: International Parallel Processing Symposium, pp. 87–93 (1997)
19. Sancho, J.C., et al.: Effective methodology for deadlock-free minimal routing in infiniband networks. In: Proceedings International Conference on Parallel Processing, pp. 409–418 (2002)
20. Singh, A., et al.: Jupiter rising: a decade of Clos topologies and centralized control in Google's datacenter network. In: ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, pp. 183–197. ACM, New York (2015)
21. Vermeij, M., et al.: MonetDB, a novel spatial columnstore DBMS. In: Free and Open Source for Geospatial (FOSS4G) Conference, OSGeo (2008)
22. Vignéras, P., Quintin, J.N.: The BXI routing architecture for exascale supercomputer. J. Supercomput. **72**(12), 4418–4437 (2016)

23. Zahavi, E.: Fat-tree routing and node ordering providing contention free traffic for MPI global collectives. J. Parallel Distrib. Comput. **72**(11), 1423–1432 (2012)
24. Zahid, F., et al.: A weighted fat-tree routing algorithm for efficient load-balancing in infini band enterprise clusters. In: 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, pp. 35–42, March 2015
25. Zilberman, N., et al.: NetFPGA SUME: toward 100 Gbps as research commodity. IEEE Micro **34**(5), 32–41 (2014)