Receive-Side Notification for Enhanced RDMA in FPGA Based Networks *

Joshua Lant[1], Andrew Attwood[1], Javier Navaridas[1], Mikel Lujan[1], and John Goodacre[1]

University of Manchester, M13 9PL, UK http://apt.cs.manchester.ac.uk/
{joshua.lant,andrew.attwood,javier.navaridas,
mikel.lujan,john.goodacre}@manchester.ac.uk

Abstract. FPGAs are rapidly gaining traction in the domain of HPC thanks to the advent of FPGA-friendly data-flow workloads, as well as their flexibility and energy efficiency. However, these devices pose a new challenge in terms of how to better support their communications, since standard protocols are known to hinder their performance greatly either by requiring CPU intervention or consuming too much FPGA logic. Hence, the community is moving towards custom-made solutions. This paper analyses an optimization to our custom, reliable, interconnect with connectionless transport—a mechanism to register and track inbound RDMA communication at the receive-side. This way, it provides completion notifications directly to the remote node which saves a round-trip latency. The entire mechanism is designed to sit within the fabric of the FPGA, requiring no software intervention. Our solution is able to reduce the latency of a receive operation by around 20% for small message sizes (4KB) over a single hop (longer distances would experience even higher improvement). Results from synthesis over a wide parameter range confirm that this optimization is scalable both in terms of the number of concurrent outstanding RDMA operations, and the maximum message size.

Keywords: FPGA · Transport Layer · Micro-Architecture · Reliability.

1 Introduction

The use of FPGAs as the main compute element within HPC systems is becoming very attractive, as we are seeing burgeoning demands from communication-intensive workloads such as AI. These workloads are much better suited to FPGA based architectures as they rely on data-flow style processing [10]. One of the key issues towards the uptake of FPGAs for HPC is the need to truly decouple the FPGA resources from the host CPU [15, 1]. This way, the FPGA will be able to communicate with other FPGA resources directly, rather than having to initiate transactions via the CPU, which will dramatically reduce the latency

* This work was funded by the European Union's Horizon 2020 research and innovation programme under grant agreements No 671553 and 754337.

of communications and better facilitate data-flow style processing among FPGAs. In theory this is relatively simple, and IP cores are available and can be readily used to provide Ethernet based communications within the fabric of the FPGA [17]. Unfortunately these existing solutions are unsuited for HPC, due to the requirements for high reliability in the network. Packet dropping is simply not an option within HPC environments, as guarantee of delivery is required. Leveraging these IPs with a TCP stack is not really feasible since it would require either either a software implementation (running in the CPU) or a full hardware-offloaded solution. The former is antithetical to our requirement that the FPGA acts as an independent peer on the network. The latter is also inappropriate due to high resource consumption and limited scalability due to its connection-based nature. In prior work [8, 2] we discussed in greater detail why traditional network protocols are unsuited for FPGA-based HPC systems, and presented a Network Inteface (NI) to enable FPGA based communication using RDMA (Remote Direct Memory Access) and NUMA (Non-Uniform Memory Access) type communications over a custom HPC network protocol. Our NI is leveraged along with our custom FPGA-based switch design [2], which lowers area and power overheads by means of a geographic addressing scheme.

This main contribution in this work is the presentation of a micro-architectural design which provides a significant enhancement in the architecture over the preliminary RDMA infrastructure presented in [8]. RDMA is a technique for transferring data to remote nodes which frees the CPU to perform useful work while network transactions are in progress, and is supported in the majority of high performance interconnection networks today. We enhance the performance of receive operations in our system by tracking incoming RDMA transfers in order to provide a receive side notification upon completion. Thus avoiding the round trip latency for the ACK, required for sender-side notifications. We show results of a send and receive operation using varying message sizes and show that the latency of small messages can be improved significantly. Our results show that we are able to scale the mechanism out to a large number of outstanding DMA operations, and achieve a latency reduction of up to 20% on small RDMA operations over a single hop distance.

Our mechanism is able to handle out-of-order packet delivery, maintaining a fully connectionless (datagram based) approach to the transport layer, and enabling the use of fully adaptive routing at packet level granularity within the network. A connectionless approach is essential to provide full decoupling of CPU and FPGA resources. Managing connection state information and the associated retransmission buffers is complex [9]. This is prohibitively expensive to implement within the FPGA fabric, given that the amount of Block RAM is limited (around 35Mb on the Zynq Ultrascale+ [20]). This is particularly true in a HPC context where the number of outstanding connections may be very large. This is the main reason why reliability is typically offered as a software solution; because the complexity of offloading is too great when rapid connection setup/teardown is required, especially for large number of concurrent connections. We argue for a connectionless approach for just this reason, to reduce the area overhead of

the transport layer, and increase the scalability by reducing the information required in the NIC. For example, we need no retransmission buffering, and push the responsibility for flow control into the network. As well as this, having the ability to route packets adaptively (as our switch design does [2]) presents the opportunity for much better load balancing within the network and enhanced fault tolerance due to the ability to properly utilize path-diversity [3].

2 Related Work

Our earlier work [8] has shown that traditional protocols such as Ethernet and Infiniband are unsuitable for use in FPGA based HPC systems, due to performance and area concerns respectively. We therefore propose the use of a custom protocol in order to avoid some of the issues with traditional networking stacks. Likewise, the majority of solutions for offering reliable communications in FPGAs are also unsuitable for our needs. This is because they typically rely on software mechanisms to enable retransmission, or hold connection states. We argue that a connectionless approach is necessary in order to enable accelerators to communicate directly with one another without CPU involvement (a key requirement for harnessing the potential of FPGAs within a HPC context [15]), and that hardware offloading of the whole transport mechanism is the only way to achieve this.

There are several FPGA based TCP-offload engines available commercially such as [11] and [12]. TCP offloading aims to either offload fully or partially the functionality of the TCP protocol into hardware. They are often touted as a good solution to the performance issues associated with the TCP/IP software stack. (These problems being latency issues due to excessive memory copying and context switching etc.) However, the TCP stack is very complex, and as such fully offloading the transport layer to hardware is very difficult, particularly for FPGA implementations. The majority of solutions therefore only offload portions of the stack to hardware such as checksumming or segmentation. To our knowledge, the only fully hardware offloaded solutions for FPGA are used for financial trading. These systems are latency-critical so the solution is fully offloaded at the expense of dramatically reduced scalability [11, 12]. Obviously this is inappropriate in the context of HPC. In [14] a solution is proposed to overcome this scalability issue, allowing for over 10,000 simultaneous connections. However, this connection based approach still suffers massive memory utilization. They require external session buffers in DRAM, amounting to 1.3GB for 10,000 sessions. Without a huge dedicated RAM for the offload engine this is extremely wasteful in terms of both memory usage and memory bandwidth.

The Infiniband specification defines a reliable, connectionless transport [7], but there is no actual hardware implementation. Grant et al. [5] propose a scheme for performing RDMA transfers using "Unreliable Datagrams" in Ethernet or Infiniband networks. They propose a method of using existing structures present in the iWARP protocol [13], writing the incoming RDMA to memory as normal at the receiver, but recording the incoming datagrams and posting to a *completion*

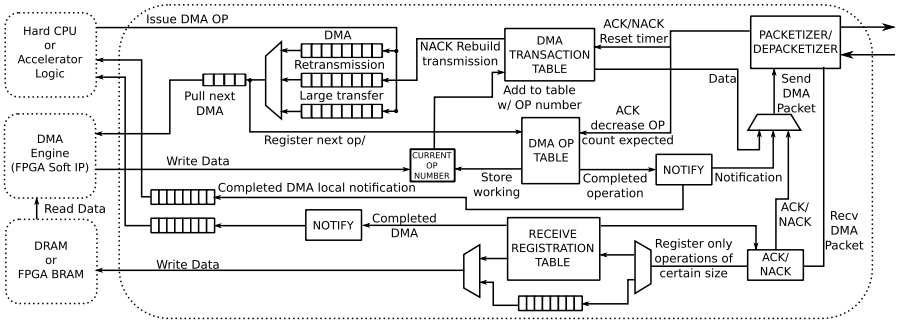

Fig. 1. Architecture of the transport layer for RDMA communications within our custom NI.

queue, which indicates that segments of a full RDMA operation have completed successfully. Their solution eliminates much of the network stack processing but is implemented in SW and does not consider reliability.

A similar approach to ours is presented by Xirouchakis et al. [21]. It describes the design of a system composed of a virtualized RDMA engine and mailboxes. This features several key differences in design from our own. They do not describe a method to store and retransmit shared memory operations as we do in [8]. They rely on software-based retransmissions, meaning that accelerator logic within the FPGA fabric is incapable of talking directly to the NI without CPU involvement. While the authors target user-level initiation of transfers to avoid the TCP/IP stack overheads, they still use a connection based approach, and only allow for multiple paths to be taken at the granularity of blocks forming these connections, not fully adaptive multipath routing and support for out-of-order delivery at the packet level as we do [8].

3 Implementation

Figure 1 shows the architecture of our hardware-offloaded transport mechanism for RDMA transfers. It can be seen here that both FPGA based accelerator logic and the hard CPU are able to utilize the NIC, issuing commands and pushing data to the network in exactly the same manner. The NIC provides reliable transmissions and allows for out-of-order packet reception using a connectionless approach. This is split into two major separate control and data-paths, one for the sending side and one for the receiving side. On the send side the CPU/accelerator issues DMA operations which are then pulled by the DMA engine from the command queues. The DMA engine is currently the Xilinx CDMA IP [19], running in Scatter-Gather mode. Every new operation which is pulled by the DMA engine is logged in the DMA OP Table in the NI. This table issues an OP number for every packet within the operation which is sent to the network and returned in the acknowledgement, keeping a count of the number

of successful transfers in a given operation. Individual packets are tracked in the DMA Transaction Table. This keeps a timeout for individual packets, and builds retransmission operation entries in the event of lost/timed out or negatively acknowledged packets. Notification of completion is given locally, to let the processor know that a DMA operation has finished sending data, and remotely, to tell the remote processor that it has new data at a given location.

3.1 Segmentation

Due to our connnectionless approach and out-of-order delivery of packets, the receiver needs to know when a whole operation is finished before it can begin work on the data, as it cannot guarantee which data has arrived at which point. Due to the importance of overlapping computation and communication for many data-intensive workloads [4] we attempt to ameliorate the additional latency that this imposes on the system by allowing for segmentation of large RDMA transfers into multiple smaller ones (as in Figure 2). Doing this is simple as the RDMA commands issued by the CPU/accelerator pass through the NI, and information can be captured and altered at this point. Figure 3 shows how the segmentation mechanism works. If a given command is seen to be over a certain threshold size it can be sent to a special "Large Transfer" queue. In this instance when the command is processed it can be assigned a status flag in the DMA Operation table. When an operation completes with this special status flag then no local notification is posted; only the notification to the receiver.

If the command is of size M, and the segment size is N, then the head of the "Large Transfer" queue remains in place for M/N operations. The offset for the base address and the number of bytes to be transferred are simply updated at the head of the queue following a new segmented command being pulled by the DMA engine (Figure 3). Upon the issue of the last command, the special status flag remains deasserted, so local completion notification for the original full transfer can be formed. The threshold for the optimal maximum size of a segmented operation is highly dependent on the structure of the application and so should be defined by the programmer during configuration of the NI.

There is little overhead in actually performing these modifications to the DMA commands. The DMA engine is set up in a cyclic buffer mode so it simply posts read requests to the NIC to pull new operations into the engine. The only difference in this instance is that the DMA engine will see a modified work item from that which the CPU/accelerator posted to the NI. Since the DMA engine can queue the next command to be performed internally, the new modified command can be formed while the current operation is in flight, so no additional latency is caused by this mechanism.

3.2 Receiver Registration

To reduce latency we track the receive side RDMA operations to provide local completion notification, and to handle out-of-order packets. Upon receiving an RDMA transaction, the operation can be logged in the Receive Registration

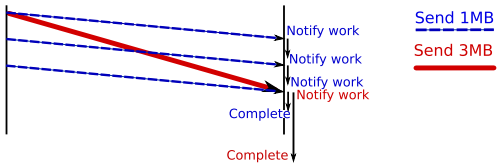

Fig. 2. Overlapping communication and computation by segmenting the DMA operations into smaller ones.

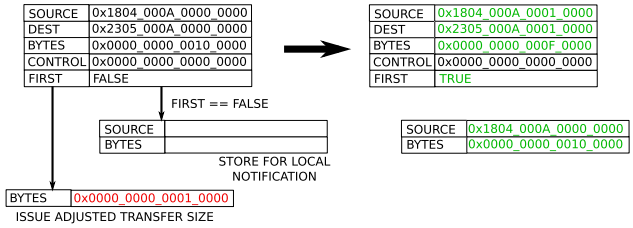

Fig. 3. Shows the update of the "Large Command" queue head for an operation of 1MB, to be split into 64K transfers.

Table (see Figure 1). It may or may not require registration depending on the transfer size (this will be discussed further in Section 4). Operations are registered by using a special entry in the "Type" field in the packet header, which is given to the first transfer of an operation. When the receiver sees this transaction they register the range of addresses which are expected from the whole operation.

Out-of-order packet delivery is handled here by creating an escape channel for any packets which currently have not had a table entry created. Until the first packet has arrived any out-of-order packets which arrived first will be put in the escape channel in order not to stall the pipeline. We are able to do this because the data that enters the NI is written to memory in a store-and-forward fashion. The data cannot be allowed to enter into memory until a CRC has confirmed the validity of the packet, so there is an (X cycles) latency corresponding to the number of flits within the packet. In this time we are able to drain the previous packet into the escape channel.

Once the first packet associated with the DMA is seen, registration of the operation is completed and a mask is used to determine when all corresponding packets have been received for the operation, and to drop duplicates. An initial mask is required to account for the fact that an operation may be smaller than the maximum possible registered operation ($Maskbitwidth \times Packetsize$). This mask is created by a barrel shifter which uses a field in the header of the first packet of the operation, which denotes the number of expected packets. We shift in zeroes to form the appropriate initial operation state. A single 1 is added to the end of this process (as the first packet must have arrived to begin this registration process). For example, if we assume a 4KB operation, a 16KB

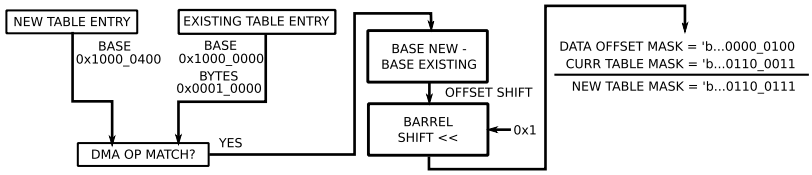

Fig. 4. Creating the bitmask for the new dma data, to check for duplicates and completion status.

mask size and a packet size of 512B, the initial mask after registration would be 'b1111_1111_1111_1111_1111_1111_0000_0001 (8 packets are needed and the first one is received).

3.3 Receiver Notification and Duplicate Data

Every time a new packet arrives the table is checked in order to determine whether any existing entry matches with the new packet. This is done by calculating whether the incoming destination is within the range of the entry currently being checked. If there is no corresponding entry in the table then the data is sent to the escape channel, and an associated timer is started. If this times out then the packet and its associated data is dropped. This timeout can happen for two reasons: *i)* The packet is a duplicate and the corresponding operation has already completed. The packet is rightfully dropped as the operation has completed and been removed from the table of active operations. In this case dropping the packet is safe because the previous packet must have sent a correct acknowledgement back to the sender. *ii)* The first packet in the operation is severely delayed or lost in the network, so registration never happens. In this case dropping the packet is safe because the sender will have received no acknowledgement or negative acknowledgement, and will itself time out and retransmit the packet. In the event that data is found to correspond to an entry in the table, but is a duplicate, the data can be safely dropped straight away and there is no need for the timer.

Figure 4 shows how the mask is updated upon receiving a new packet. If the table entry being checked is found to match the incoming data, then it proceeds to create a new mask. The base address of the entry and the number of bytes of the operation are used to calculate whether the operation being checked in the table relates to the new packet arriving. An offset is then created for a barrel shifter, which generates a mask to cause a bit flip. If the mask is found to be all 1's then the operation must be completed. If $Newmask == Originalmask$ then the packet must be a duplicate and can be dropped.

Once a full operation has been completed the receiver is notified locally of the RDMA operation, which saves a full round-trip packet latency compared to waiting for the sender to provide a completion notification upon receiving the last acknowledgment (see Fig. 5). The notification is currently sent to a queue which

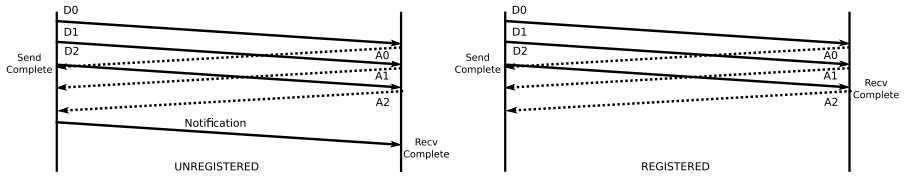

Fig. 5. Time for send and receive operations to complete for registered and unregistered transfers.

can be polled by the receiver. Doing this allows for a non-blocking or a blocking receive to be performed in the application. Polling the notification queue when empty returns a special default value. This can be used to effectively create a spin-lock in the software, only returning from the blocking receive function when the notification has indeed been posted, and the return value is not equal to the default value.

3.4 Non-Registered Operations

There may be points where either registering an operation is unnecessary, or is not sensible given the number of communications (for example in a many-to-one collective involving many nodes). In this case the operation remains unregistered and we must suffer the additional round trip latency for acknowledgement. However, in this case there is no need to track and handle duplicate packets, or out-of-order delivery. The addresses of the packets which form the DMA operations are simply memory locations within a global virtual address space (described in [6]), it does not matter if this memory location is overwritten, because the acknowledgement for the operation happens only once all the corresponding packets have been acknowledged to the sender. We provide strong ECC protection for ACK packets so that they will only be lost or corrupted in the most exceptional circumstances. If packets arrive out-of-order then they are simply written to the correct place in memory regardless, as the packet has a base address associated with it, which is formed in the DMA engine at the sender.

4 Evaluation

The Network Interface, and thus all the components shown and discussed in Section 3 are implemented completely within the fabric of the FPGA. For all evaluation within this Section we use the Xilinx Zynq Ultrascale+ ZCU102 development board (part number EK-U1-ZCU102-G). The test setup is shown in Figure 6. There are two entirely segregated datapaths within the FPGA, emulating completely the action of a distributed setup except we implement the send node's IP and the receiving node's IP within a single FPGA. We have shown in previous work communication over a complete networked solution including the router/switch [2, 8], but in order to more accurately measure time,

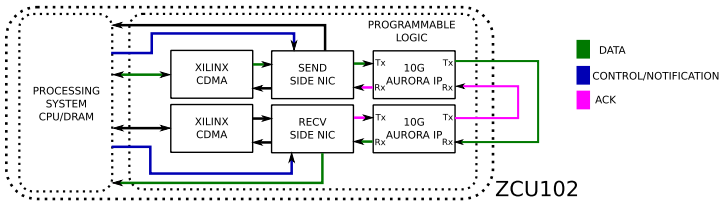

Fig. 6. Experimental setup on the Zynq Ultrascale+ FPGA.

we use a loopback setup. Using the NIC in conjunction with the switch allows for a much higher number of connections to/from a single node. The processing system contains four hard-core ARM-A53 processors, as well as cache-coherent interconnect, IOMMU, DRAM controller etc, while the programmable logic is just that; the FPGA fabric of the device [18]. It can be seen that the current DMA engine is the Xilinx CDMA [19], and we use Aurora PHY IP to enable 10G line-rate across the links [16]. This IP is used simply to perform serialization and 64/66b line encoding/decoding, and does not wrap the packet using any other protocol. The frequency of all components implemented within the FPGA fabric is 156.25MHz. The processing system runs at 1GHz.

4.1 Latency of Send and Receive Operations

In order to balance the requirements for low latency transfers with reduced area overheads, only a limited range of message size for registration is required. Figure 7 shows the results of an experiment to show the performance benefits of registered receive side transactions. This shows the latency for the transfer of data and notification of completion in a user-space application for a single hop transfer. The latency of the send operation is the time taken to configure the DMA engine from user-space, and for notification to be received at the sender that the DMA engine has pushed all the data into the network. The measurement we take is thus for a non-blocking send, for the data to simply enter the network. A blocking send operation would have higher latency than the registered receive operation since it must wait for the last ACK to arrive. The latency of receive operations are measured from the time the sender begins to initialize the transfer, until the receiver gets notification that the DMA data is placed in memory, either by local notification from the NI (Registered), or as a notification packet from the sender.

As shown in Figure 7, the latency of a receive operation for a 1KB transfer is around 5.23μs, and for a registered receive is only 4.21μs, cutting $\approx$20% from the latency of the recevie side being notified of the transaction. We also see that the performance gains from this registration technique diminish with transfer size and become insignificant at around 32KB. At much larger transfers the measured latency for send/recv/registered recv are very similar, as is seen in the convergence of the results in Figure 7. This is because the extra round trip

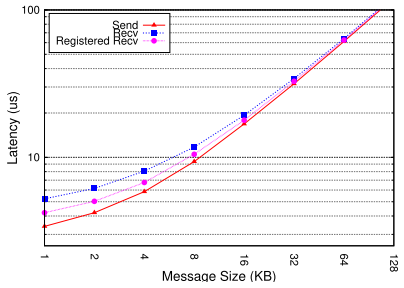

Fig. 7. Latency of a send/recv operation over a single hop distance.

latency is dwarfed by the overall transfer time for the data. What this means in practice is that registered transactions will only show significant benefits within a certain range of smaller message sizes. Although this is dependent on the distance from the destination and the network load (affecting latency). As the distance between source and destination increase, or the load of the network goes up, we would see larger and larger message sizes be able to benefit from receive side registration. The distance between sender and receiver can be worked out easily owing to the geographical routing scheme which we employ [2], so adjusting the threshold for registration based upon this would be trivial. However, dynamically adjusting these thresholds based upon the network load may be very difficult and be potentially very wasteful of resources.

4.2 Area of Receiver Registration Module

Clearly there will be high variability in the performance gains of registering the receive side operations, depending on the distance of communications. It therefore seems appropriate to perform a parameter sweep for various configurations of number of simultaneous outstanding operations the node can handle, and the

Table 1. Area utilization (% total) for various combinations of max packet size and table depth. Total LUTs = 274080, total BRAMs = 912.

		Bitmask Vector Size									
		64 (32KB)		128 (64KB)		256 (128KB)		512 (256KB)		1024 (512KB)	
		LUT	BRAM	LUT	BRAM	LUT	BRAM	LUT	BRAM	LUT	BRAM
Table Size	64	2230 (0.81)	47 (5.15)	2936 (1.07)	49 (5.37)	4286 (1.56)	53 (5.81)	6589 (2.40)	60 (6.58)	11072 (4.04)	74 (8.11)
	128	2282 (0.83)	47 (5.15)	2942 (1.07)	49 (5.37)	4289 (1.56)	53 (5.81)	6501 (2.37)	60 (6.58)	11092 4.04)	74 (8.11)
	256	2266 (0.82)	47 (5.15)	2973 (1.08)	49 (5.37)	4366 (1.59)	53 (5.81)	6842 (2.49)	60 (6.58)	11124 (4.05)	74 (8.11)
	512	2298 (0.84)	47 (5.15)	2974 (1.08)	49 (5.37)	4367 (1.59)	53 (5.81)	6843 (2.49)	60 6.58)	10965 (4.00)	74 (8.11)
	1024	2273 (0.83)	47 (5.15)	2976 (1.09)	52 (5.70)	4363 (1.59)	57.5 (6.30)	6575 (2.39)	68 (7.45)	11088 (4.04)	89.5 (9.81)

largest possible size of operation for receiver registration. Table 1 shows the area utilization of the *Receive Registration* module (shown in Figure 1), under differing configurations. We consider bitmasks for between 32KB and 512KB and packets of 512KB–a small packet size used to ease congestion and help with load balancing. We vary the number of outstanding operations (table entries) between 64 and 1024.

The results show that varying the maximum operation size for registration has little effect on the number of LUTs. This is because the logic to decode/encode the mask is not significant, compared with other components in the module. The number of BRAMs jumps considerably at certain boundaries, which is due to the odd bit width of the table entries. Effectively this creates a scenario where we can gain "free" entries to the table because of the fixed size BRAMs being utilized more efficiently. It is also worth noting that the number of BRAMs for the smallest 64x64 configuration does not correspond to the utilization of the table. This is because the storage for the data in the escape channel is set to enable 64 full packets to be held in the NI. This uses 43 BRAMs, which is why we still see a baseline for the BRAM utilization at this small configuration. Although this value is highly acceptable, and not prohibitive for the implementation of accelerators in combination with our NI, with the largest possible configuration only requiring 10% of the total BRAMs, and uses no DSP slices, which are key for efficient floating point arithmetic

5 Conclusions

In this paper we have presented an optimization for the hardware-offloaded transport layer of an FPGA based Network Interface. A micro-architecture is presented which allows for the receiver of an RDMA operation to register the operation, thereby enabling receive side notification upon completion of the operation. We show that for small RDMA operations the latency of the receive operation can be reduced by ≈20%. This can be leveraged with a method of segmenting large DMA operations into a number of smaller ones, thereby enabling us to maintain a connectionless (datagram based) approach to our transport layer, while allowing communication and computation to overlap. The connectionless approach maintains scalability of the system, and allows for fully adaptive routing at packet level granularity, giving better load-balancing properties to the network.

We provide an analysis of the area utilization of various configurations of the receive-side registration module, and show that, due to the fixed sized BRAMs and the odd bit-width of table entries, certain configurations make better use of the BRAMs. In the most aggressive implementation, the total BRAM use of the receive registration module is below 10% of the available, whereas the number of LUTs is around 4%. More reasonable configurations lower these to around 6% and 1.5%, respectively. Hence, the overall area utilization is very acceptable, leaving plenty for use by accelerator blocks etc. Particularly when noting that our implementation does not utilize any DSP blocks on the FPGA.

References

1. Caulfield, A.M., Chung, E.S., Putnam, A., Angepat, H., Fowers, J., Haselman, M., Heil, S., Humphrey, M., Kaur, P., Kim, J.Y., et al.: A cloud-scale acceleration architecture. In: The 49th Annual IEEE/ACM International Symposium on Microarchitecture. p. 7. IEEE Press (2016)
2. Concatto, C., Pascual, J.A., Navaridas, J., Lant, J., Attwood, A., Lujan, M., Goodacre, J.: A cam-free exascalable hpc router for low-energy communications. In: International Conference on Architecture of Computing Systems. pp. 99–111. Springer (2018)
3. Dally, W.J., Aoki, H.: Deadlock-free adaptive routing in multicomputer networks using virtual channels. IEEE transactions on Parallel and Distributed Systems **4**(4), 466–475 (1993)
4. El-Ghazawi, T., El-Araby, E., Huang, M., Gaj, K., Kindratenko, V., Buell, D.: The promise of high-performance reconfigurable computing. Computer **41**(2) (2008)
5. Grant, R.E., Rashti, M.J., Balaji, P., Afsahi, A.: Scalable connectionless rdma over unreliable datagrams. Parallel Computing **48**, 15–39 (2015)
6. Katevenis, M., Chrysos, N., Marazakis, M., Mavroidis, I., Chaix, F., Kallimanis, N., Navaridas, J., Goodacre, J., Vicini, P., Biagioni, A., et al.: The exanest project: Interconnects, storage, and packaging for exascale systems. In: Digital System Design (DSD), 2016 Euromicro Conference on. pp. 60–67. IEEE (2016)
7. Koop, M.J., Sur, S., Gao, Q., Panda, D.K.: High performance mpi design using unreliable datagram for ultra-scale infiniband clusters. In: Proceedings of the 21st annual international conference on Supercomputing. pp. 180–189. ACM (2007)
8. Lant, J., Concatto, C., Attwood, A., Pascual, J.A., Ashworth, M., Navaridas, J., Luján, M., Goodacre, J.: Enabling shared memory communication in networks of mpsocs. Concurrency and Computation: Practice and Experience (CCPE) (2018)
9. Mogul, J.C.: Tcp offload is a dumb idea whose time has come. In: HotOS. pp. 25–30 (2003)
10. Ovtcharov, K., Ruwase, O., Kim, J.Y., Fowers, J., Strauss, K., Chung, E.S.: Accelerating deep convolutional neural networks using specialized hardware. Microsoft Research Whitepaper **2**(11) (2015)
11. PLDA: An implementation of the tcp/ip protocol suite for the linux operating system (2018), https://github.com/torvalds/linux/blob/master/net/ipv4/tcp.c
12. Intilop Corporation: 10 g bit tcp offload engine + pcie/dma soc ip (2012)
13. Ohio Supercomputing Centre: Software implementation and testing of iwarp protocol (2018), https://www.osc.edu/research/network_file/projects/iwarp
14. Sidler, D., Alonso, G., Blott, M., Karras, K., Vissers, K., Carley, R.: Scalable 10gbps tcp/ip stack architecture for reconfigurable hardware. In: Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on. pp. 36–43. IEEE (2015)
15. Underwood, K.D., Hemmert, K.S., Ulmer, C.D.: From silicon to science: The long road to production reconfigurable supercomputing. ACM Transactions on Reconfigurable Technology and Systems (TRETS) **2**(4), 26 (2009)
16. Xilinx Inc.: Aurora 8B/10B (10 2016), v11.0
17. Xilinx Inc.: 10G/25G High Speed Ethernet Subsystem (4 2017), v2.1
18. Xilinx Inc.: Zynq Ultrascale+ Device, Technical Reference Manual (12 2017), v1.7
19. Xilinx Inc.: AXI Central Direct Memory Access (4 2018), v4.1
20. Xilinx Inc.: Zynq UltraScale+ MPSoC Data Sheet: Overview (11 2018), v1.7
21. Xirouchakis, P., et al.: The network interface of the exanest hpc prototype. Tech. rep., ICS-FORTH / TR 471, Heraklion, Crete, Greece (2018)