CusComNet: A Customisable Network for Reconfigurable Heterogeneous Clusters

Stewart Denholm, Kuen Hung Tsoi, Peter Pietzuch and Wayne Luk
Department of Computing, Imperial College London, UK
{swd10, khtsoi, prp, wl}@doc.ic.ac.uk

***Abstract*—Computer clusters equipped with reconfigurable accelerators have shown promise in high performance computing. This paper explores novel ways of customising data communication between accelerator nodes, which is often a bottleneck when scaling up the cluster size. Based on the direct connection of high speed serial links between advanced reconfigurable devices, we develop and evaluate CusComNet, a scalable, flexible and efficient communication framework. The CusComNet framework is built around customisable, packet-based communication and supports three main types of customisation: packet protocol customisation, system-level customisation, and prioritised communication customisation. A performance model for estimating CusComNet's communication latency is proposed and demonstrated. Our framework is applied to a 16-node cluster, each node of which contains an FPGA accelerator which can be connected directly to other FPGA accelerators. The proposed framework can be used to improve the scalability of a reconfigurable cluster by involving more nodes in a single application. Performance measurements show high efficiency data throughput for both large and small data volumes, as well as low communication overhead.**

***Keywords*-FPGA; reconfigurable heterogeneous clusters; customisable networks**

I. Introduction

The heterogeneous reconfigurable cluster combines ideas from both computer clusters and hardware accelerators. In such clusters, both the algorithm kernels are implemented in reconfigurable hardware resources such as Field Programmable Gate Arrays (FPGAs) for optimised performance, and the application is mapped to multiple accelerators for parallel computation. Many clusters employ a standard network, such as Gigabit Ethernet, to connect together its nodes; this network can become a bottleneck. Fortunately some accelerators contain network interfaces, enabling them to connect directly to one another. So how can we make the best use of this reconfigurable communication capacity?

Before we address this question, let us look at an example. Figure 1 shows the paths when a block of data is transferred from one accelerator to another. As indicated by the thin arrow lines in the figure, transferring data between accelerators through Ethernet must go via the PCIe channel and host memory bus multiple times. Alternatively, since there are fast, point-to-point (P2P) serial communication macro-blocks in modern FPGA devices, we can make use of these communication resources, as indicated by the thick arrow lines. By eliminating data movement during the accelerator-host communication and host software stack overhead, the communication overhead can be significantly reduced and thus achieve better performance in applications. When communication is the limiting factor of the cluster scalability, we can expect application performance to improve further by the addition of more accelerators.

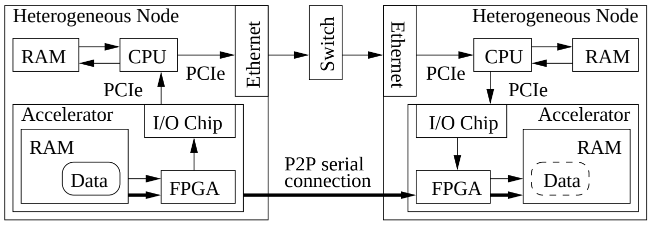

Figure 1. Data transfer between accelerators in a heterogeneous cluster.

There are several challenges related to heterogeneous communication in cluster computing. The difficulty lies in providing a systematic and efficient framework to benefit the communications within real-world applications. This requires us to first understand the characteristics of the dedicated P2P direct connections between the FPGA accelerators. It is also important to make the framework customisable so that the final design can be specifically optimised for a given application. To support message routing without an external network switch, hub-based routing facilities must be available. To assist the development of designs at a higher level, analytical models and application wrappers are necessary to improve ease of use. In the following, we address these issues and evaluate the proposed approach. The new contributions of our work include:

- CusComNet, a customisable communication framework for heterogeneous clusters, based on the fast serial connections on FPGA devices. The data format, connection schemes and architecture parameters can be optimised for specific applications to improve system performance.
- A performance model for exploring the effects of customisation in CusComNet. System parameters such as bandwidth and latency are captured in the model to provide performance estimation.
- Implementation of CusComNet on a production cluster

system, and evaluation of the proposed performance model against the measured bandwidth and latency values. This is done across directly connected nodes and communications requiring routing via one or more intermediate nodes. An N-Body simulation shows that the overall performance can be improved by about 3 times with over 12 nodes before saturation.

The paper is organised as follows. Section II reviews systems with a dedicated inter-FPGA network. Section III presents the details of the network architecture at various levels. Section IV presents an analytical model for performance estimation when utilising the proposed framework. Section V describes the details of a CusComNet implementation within a 16-node cluster. Section VI presents the results of experiments carried out on our implementation. Finally, Section VII draws conclusions.

II. Background

Various FPGA-based clusters have been reported in the last few years. The following presents several examples, and describes the differences between our work and theirs.

The Cray XR1 blade [1] for the XT5 system contains two optional Xilinx Virtex-4 XC4VLX200 FPGAs per blade. Each FPGA is interfaced to the CPU through a HyperTransport channel. Inter-FPGA communication in the XT5 system must go through the CPU and the Cray proprietary SeaStar2+ interconnection chips. Software running on the CPU must be developed first to make use of the inter-FPGA communications.

The SGI RC100 blade [2] contains two Xilinx Virtex-4 XC4VLX200 FPGAs. An SGI proprietary NUMAlink ASIC chip is used to connect different blades with over 6Gbps bandwidth. SGI provide the core service IPs for FPGA development. At the cost of one-tenth of the FPGA logic resources, the core service provides connection to both NUMAlink and external memory modules. SGI also provides an abstraction layer which enables one application to be spanned across multiple FPGA devices. This abstraction layer is comprised of library function calls running on the CPU, hence inter-FPGA communications have to go through the CPU.

The *Maxwell* system [3] has 32 computing nodes hosted in five IBM BladeCenter chassis. Two FPGA accelerators using Xilinx Virtex-4 FPGAs are hosted on each node. Besides the Gigabit Ethernet on the host side, there is an 8×8 2D torus network using the MGT serial ports of the FPGAs. Each FPGA accelerator is connected to four other accelerators using the single channel HSSDC2 InfiniBand cable. To simplify development, the project excludes routing logic in the FPGA and thus users must develop their own mechanisms for transferring data to the indirectly connected FPGA accelerators.

The Reconfigurable Computing Cluster (RCC) project developed a prototype [4], called *spirit*, with 64 Xilinx ML410 evaluation boards. The PowerPC in the Virtex-4 FPGA is used as the main CPU in the cluster and the connection between FPGA boards is through SATA cables. The SATA channel provides up to 2.5Gbps bandwidth. Extensive experiments have been carried out to test the communication performance against various factors including cable length. However, there is little information about applications supported by this cluster.

Our research complements previous work in that we focus on opportunities for customising communication networks in heterogeneous clusters based entirely on extending the FPGA configuration, without dedicated or proprietary hardware extensions. To estimate the effects of customisation we provide a performance model, and demonstrate our approach using a realistic application based on N-body simulation.

III. Communication Framework

While much has been written about ways of customising computation and network-on-chip, customising communications within computing clusters appears less popular. We aim to provide a unified, easy to use inter-FPGA communication network capable of both specific customisations for user applications, as well as application-independent packet routing at a lower level. The purpose of our research is to:

- determine the areas of cluster communications that benefit from customisation;
- provide a framework that facilitates multiple applications communicating within a single cluster;
- quantify the performance gain for customised communications.

The following presents a systematic exploration of customising communications for an FPGA-based cluster, focusing on the data link layer and the network layer. A prototype implementation of CusComNet on a 16-node heterogeneous computing cluster is also outlined.

A. Customisation: Overview

CusComNet supports customisation in three main areas: packet protocol customisation, system-level customisation, and prioritised communication customisation.

Packet protocol customisation: Compared to traditional Ethernet-based communications, the direct FPGA-to-FPGA connections allow larger customisation benefits—i.e., more direct control over the communication stack—achieved via user definable packet and buffer sizes, priority levels, etc. FPGA resources which remaining available after implementing the user application can easily be incorporated into the communication framework to improve system performance, for example, by increasing buffer depths or packet data widths. Application-specific improvements can be made by exploiting the run-time flexibility of the packet protocol. For example, the upper packet size limit can be large, but the actual sizes of transmitted packets can vary between, and within, each application.

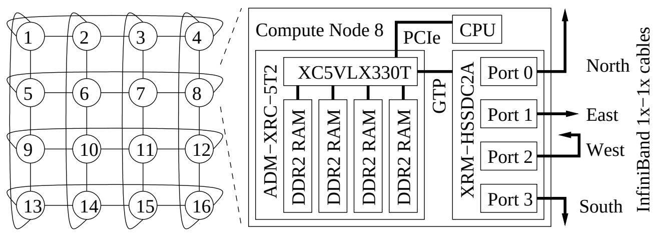

Figure 2. A prototype CusComNet within a 2D torus network utilising P2P FPGA serial communication.

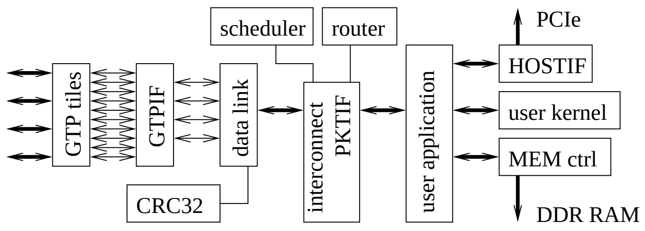

Figure 3. Design architecture utilising the GTP core.

System-level customisation: Build-time customisations allow for application and topology specific improvements via the scheduling algorithm for incoming data and the packet routing functionality. Nodes can be designed and optimised for specific tasks, such as those designated as communication hubs incorporating additional buffer space and separate routing algorithms. A universal routing algorithm can also be defined to implement hard-coded logical topologies, or a less stringent system can be created via preferred transmission paths within the scheduler.

Prioritised communication customisation: The inclusion of packet priorities enables the creation of communication hierarchies to allow for packets or whole applications within a cluster to be given preference over others. Cluster-wide control applications, such as those for performance measurement or load-balancing, will benefit from higher prioritised traffic.

Note that the underlying design of the CusComNet framework is application-independent and can be used by multiple cluster applications running simultaneously. This is facilitated by the low level routing allowing 'foreign' packets to pass through an application's network. Foreign packet routing is particularly important for heterogeneous clusters since different resources may be spread throughout the network, and producers and consumers may not necessarily occupy adjacent nodes.

B. *Framework Implementation*

The CusComNet framework has been implemented on a 16-node cluster in order to demonstrate its operation and measure communication performance. Each compute node has one FPGA accelerator board hosting a Virtex-5 FPGA for user applications, interfaced to the host system through the PCIe bus. There are 4 banks of DDR2 memory, each with 512MB capacity, associated with the Virtex-5 FPGA. Each accelerator has four RocketIO GTP transceivers exposed to the external interface, linking the accelerator to four others and forming a 2D torus network (Figure 2), similar to that in [3]. The cluster allows topologies to be customised by re-arranging the physical connections. InfiniBand cables are used as communication links between adjoining nodes and are connected directly to the FPGAs. The InfiniBand cables vary in length from 0.5 to 2.5 meters and are rated for a maximum bandwidth of 2.5 Gbps. Beside the FPGA, each cluster node has a CPU with Ethernet access. An Ethernet network in a switched star topology connects each node via an Ethernet controller on the PCIe bus, providing an additional means of inter-node communication.

Figure 3 shows a simplified structure of our prototype design using the embedded fast serial connection cores within the FPGA. The framework is implemented as a multi-layered wrapper around the four physical GTP tiles, with the functionality split into data link and network layers.

C. *Data Transmission*

The lowest layer of CusComNet is the data link layer which is specific to each transceiver on a node's FPGA; in the case of our prototype design, the four GTP tiles. We provide error checking at this level, as packets pass from one node to the next, rather than only at the packet's final destination. This allows us to gather information about the status of each node-to-node connection as well as ensure the integrity of the packet at each communication step. Through analysis of any errors caused by transmitter and receiver misalignment or altered transmission data, as well as by counting the frequency of repeated transmissions, it is possible to create an overview of the network's state at run-time. Future work will make this information available to network routing modules, allowing the creation of algorithms that can dynamically alter network transmission paths to avoid error prone connections.

Data are transmitted using the 8B/10B encoding [5] scheme with special characters used to denote the IDLE, SYNC, SOF (start of frame) and EOF (end of frame) symbols. We support full duplex communication, enabling simultaneous data transmission and receiving. During packet transmission, the transmitter first syncs the channel with the receiver then transmits the serialised bit stream, enclosed by the SOF and EOF symbols. Finally, the locally calculated CRC is sent, which the receiver compares to its own calculated value, prompting either the confirmation of a successful transmission, or a request to resend the packet.

CusComNet supports variations in clock speed, synchronisation period and frame acknowledgement period. Through experimentation, we observe that the maximum applicable frequency for the communication link is affected by cable attributes and electronic noise. The synchronisation period and frame acknowledgement timeout period are controllable through VHDL generic ports. Together with the adjustable link frequency, these customisable parameters are critical for maintaining reliable links between FPGAs. Future work will focus on the run-time optimisation of each inter-node connection's synchronisation and frame acknowledgement periods. By taking into account the different cable lengths and error rates between each node-pair, we aim to minimise the impact of any connection issues, therefore maximising the available throughput.

D. Packet Protocol Customisation

Packets are assembled and processed within the network layer. Each packet contains the unique addresses of the destination and source nodes. The bit widths of these two fields are automatically determined when the user defines the maximum number of nodes in the cluster. Packets also contain a type identifier that denotes either a data or control packet. Further types can be defined to support more complex control or data-passing mechanisms, such as congestion detection and flow control. Each packet also holds a unique identifier allowing out-of-order transmission, however, the user must manually reorder the arriving packets. By examining each packet's unique identifier, the network layer automatically rejects any duplicate packets targeted to its local node. The data field of a packet is design-time customisable. At run-time, packets can be transmitted with data widths of any size upto this value. One example is acknowledgement packets which do not carry data and so will have a data width of zero.

This layer makes use of design-time customisable FIFO buffers for storing packets. The maximum payload size of a packet must be selected by the user to reflect the most common message size of the application. The storage space in the buffer will be wasted if the maximum payload size is consistently larger than the actual size of the transmitted packets. If the packet payload size is too small, the number of generated packets will increase which, coupled with the packet's embedded header, saturates the buffer more quickly. With the combination of customisable payload sizes and maximum buffer depths, we allow a trade-off between FPGA storage resources and the blocking probability of a node. Nodes with different queue sizes can improve the overall performance of applications with irregular communication patterns.

E. System-level Customisation

The network layer provides packet switching functionality in the *PKTIF* module, allowing FPGAs to communicate with each other via intermediate nodes. In our prototype system, the 2D torus network provides each node with 5 possible inputs: the 4 GTPs and the local node itself, as shown in Figure 4. A Round-robin scheduler—isolated in a separate module—ensures an evenly distributed access probability for both routed and locally generated packets. More advanced

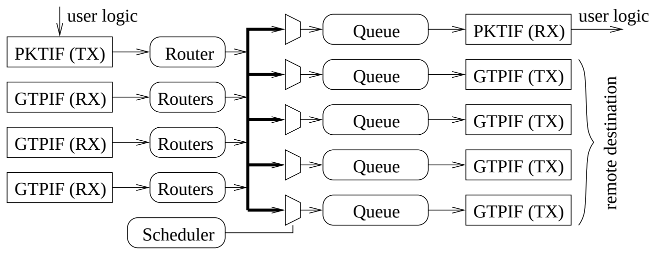

Figure 4. Network layer routing.

algorithms can be provided simply by changing the scheduler module.

A routing module is responsible for determining the destination of each incoming packet, with waiting packets held in the design-time customisable FIFO buffers. The router for the 2D torus in this work routes packets along the shortest path, based on the relative column and row positions of the local and destination nodes. Similar to the scheduler, packet routing information is held in a separate module. Other topologies or priority routing algorithms can be implemented by changing the routing module.

If the user logic occupies only a small percentage of available FPGA resources, the additional hardware resources may be used to optimise the communications via appropriate scheduler and routing module designs. Conversely, these modules may also be scaled back to meet the demands of the user logic.

F. Prioritised Communication Customisation

To support customisation of prioritised communication, a field for packet priority is included. A priority is assigned by the user to every outgoing packet. When routed by the local node and processed by any intermediate nodes, packets are stored and retrieved from FIFO buffers according to their priority level. Each FIFO buffer is defined within CusComNet as one or more separate buffers: one for each of the priority levels specified by the user. At build-time, the user is able to set the number of priority levels as well as define the depth of each corresponding buffer. This approach enables CusComNet to support a number of different prioritised packet protocols, or remove prioritisation completely to allow for a single, large storage buffer.

G. Usability

The *PKTIF* module, as shown in Figure 4, provides a wrapper for user applications, abstracting away any data link or packet level information. Application scaling has no effect on the wrapper or communication operations at the user level, as users simply provide the data, destination node and the associated priority.

When communicating between nodes using a PCIe-based system, data transmitted between FPGAs must first be passed to the CPU-level via the PCIe bus, then to the Ethernet controller for transmission. Received data are then routed via a CPU-level program before being delivered to the receiving node's FPGA. The use of a directly-connected, dedicated inter-FPGA network does not require the development and inclusion of a separate application at the CPU-level to handle communications. This way, the majority of the development focus can be shifted towards the FPGA, with the exception of FPGA initialisation. Applications that make use of both the CPU and FPGA will also benefit as they will not need to arbitrate for use of the connection channel.

IV. Performance Model

The performance of the proposed customisable communication framework depends highly on the target application and also on the physical network connection topology. While various parameters can be customised in the framework to optimise the performance for an application, it is not simple to identify suitable values before actually implementing the design. Exploring the complete design space in multiple implementation iterations can be time consuming. Thus, we develop a performance model for rapid parameter exploration in the early design stages.

To model network performance, two sets of parameters are required. The platform parameters represent the attributes of the underlying hardware designs and configurations; these parameters are obtained by measurements and profiling on a specific FPGA cluster. The application parameters represent the communication pattern of the implemented algorithm; these parameters are obtained either by analysing the algorithm or by profiling.

Table I shows the platform parameters used in our performance model. We define a *link* to be a direct connection between two FPGA accelerators and a *channel* to be a path for a packet to travel from the source FPGA to the destination FPGA. A *channel* is a connected sequence of *links*. We also define the FPGAs involved in a channel to be *hubs* if they are neither the source nor the destination. Here T_l is the FPGA link transmission delay, which covers the propagation delay of the cable and is negligible for our copper wire connection. T_r is the FPGA packet routing latency and it covers the time for unpacking, inspecting and routing the packet after it is received by the GTP tile. T_a is the latency for data moving between the user logic and the routing logic.

Experiments show that some parameters, such as B_l, are dependent on the physical environment of the cluster nodes and have a range of possible values. To simplify the problem, we use the average values of these parameters in our performance model.

Table I also shows the application parameters used in our performance model. Again, for simplicity, the average values will be used for these parameters. Here we assume that the application sends out data randomly to a random destination.

Table I
Parameters for network performance modelling.

symbol	unit	meaning	typical value
		platform parameters	
k		number of links along a path	1 - 3
p	bits	size of a packet	16 - 528
q		max. packets in a queue	64
B_l	Mbps	bandwidth of physical link	1600
T_l	μs	FPGA link transmission latency	0
T_r	μs	FPGA packet routing latency	0.57
T_a	μs	application to routing logic latency	0.13
		application parameters	
d	bits	total data size to be transferred	N/A
λ	1/s	average packet rate	N/A
$\tilde{k}$		average links per path	N/A

The total time (T) required to transmit a packet of data from one node to another can be expressed in the following equation. The first two terms represent the total latency along the path, and the last term is the time required to transmit the actual data bits:

$$T = (k-1) \times (T_l + T_r + T_{pkt}) + 2 \times T_a \tag{1}$$

where $T_{pkt} = p/B_l$ is the packet transmission time. This equation is sufficient to capture the theoretical performance of transferring a packet without queuing delay. More advanced models are needed to estimate the communication performance at the application level. This is due to link contention when multiple packets are routed to the same output link. In this case, the input packet must first wait for the scheduled time slot from the scheduler, then wait again to clear the queue before it can reach the GTP tile. Our performance model considers this situation for a more accurate estimation.

Without losing generality, we consider the behaviour of each router and queue combination using an independent M/M/1 queuing model [6]. Since the average number of packets generated per second (λ) is captured as a model parameter, and we have four outgoing GTP links per accelerator, the enqueue rate is:

$$EQ_{pkt} = \lambda/4. \tag{2}$$

Since there is little delay introduced between the packet queues and the GTP links, the dequeue rate can be derived from the packet transmission time (T_{pkt}) as:

$$DQ_{pkt} = 1/T_{pkt} = 1/(p/B_l) = B_l/p. \tag{3}$$

Here we assume that the packets addressed to the local node are consumed in zero time, and we define $\rho = EQ_{pkt}/DQ_{pkt}$. The expected number of packets in a queue is then:

$$\tilde{q} = \rho^2/(1-\rho). \tag{4}$$

The expected waiting time in the queue is:

$$T_q = \rho/(DQ_{pkt} - EQ_{pkt}). \tag{5}$$

The current Round-Robin scheduler can process a packet in one clock cycle and there are only 5 possible inputs to any queue: the four GTP links and the local node itself. So the scheduling delay, T_s, is practically zero in this case. For more advanced scheduling algorithms, the T_s value may be significant in the overall performance. Therefore the average time for a packet to pass through a hub is

$$T_p = T_s + T_q + T_{pkt}. \tag{6}$$

For the average $\tilde{k}$ links distance in a packet transmission, the average time to transfer a packet is $T_p \times (\tilde{k} - 1)$. Thus the average time spent by the application to send a packet is obtained by combining Equations 1 and 6.

$$\tilde{T} = (\tilde{k} - 1) \times (T_l + T_r + T_p) + 2 \times T_a \tag{7}$$

Note that this model is based on a set of assumptions such as a 2D torus network topologies and Round-Robin scheduler. When these assumptions are no longer valid with different platforms or different applications, the above equations must be changed accordingly. To obtain more accurate performance estimation, network simulators such as [7] should be used for each specific application.

To apply this performance model, the user should first collect the platform dependent parameters by experimentation, as shown in Section VI. This process is needed only once for each platform and implementation configuration and the results can be reused for any application. The typical values in Table I are obtained as described in Section VI for our Virtex-5 platform.

Equation 4 can help users to determine a suitable queue size—the customisation parameter in Section III-A—based on the application's communication pattern. Equation 7 provides a simple and accurate estimation of the network performance. Experiments in Section VI shows that the measured results agree with the estimated network performance using this model.

V. Implementation Details

We implement our prototype framework on Xilinx Virtex-5 LX330T FPGAs within a production cluster. The device utilisation is given in Table II, where each slice contains four Look-up Tables (LUTs) and four registers. As this is an initial design we do not focus on resource efficiency; this will become an important aspect in later design iterations. Since CusComNet must be incorporated into existing applications, it must therefore possess as small a resource footprint as possible.

Table II
CusComNet's slice utilisation on the Virtex-5 LX330T.

Slice Type	Number Utilised	Percentage of Total
All	21,219	40
Registers	47,685	22
LUTs (Total)	48,444	23
LUTs (Logic)	35,842	17
LUTs (Memory)	12,408	6

For our prototype design, we measure the throughput and latency of CusComNet's communications between adjoining FPGAs and those separated by at least one intermediate hop. The maximum packet payload size is set as 64 bytes and the FIFO buffers are able to hold 64 packets. The packet transmission time is measured using the FPGA internal clock, then converted into seconds based on the operating frequency of 100MHz. This operating frequency translates to a line speed of 2Gbps; it is set to this value since experimentation shows that, above this frequency, transmission errors are increased. Line speeds at the InfiniBand cable's maximum speed of 2.5Gbps result in communications taking up to two orders of magnitude longer than those transmitted at 2Gbps. This is mainly attributed to the transmitter and receiver frequently becoming misaligned, requiring more packets to be retransmitted.

Two experiments will be performed to determine the bandwidth of the prototype design. The first experiment involves sending small packets of data in order to measure the bandwidth when transmitting lower quantities of data. For smaller payload sizes, we do not saturate the packet buffers, but the header occupies a larger percentage of the total packet and thus has an effect on performance. The second experiment involves sending larger numbers of packets between two nodes to determine to what extent the effect of buffer saturation has on overall throughput, and to find the worst-case average bandwidth. We determine communication latency by measuring the time taken to transmit a packet containing zero bytes of data. Transmissions between non-adjoining nodes are also tested to determine to what extent the latency increases with successive hops, and its effect on the bandwidth.

VI. Results

A. Network Performance Measurement

The average bandwidth is found to be around 1479Mbps when transmitting small data packets between adjoining nodes, as shown in Figure 5. For packet payloads at or near the maximum value of 64 bytes, the bandwidth approaches the 1600Mbps limit of the 2000Mbps network connection when using 8B/10B encoding, as the amount of data in each packet becomes large relative to the header size. Figure 6 shows that as the number of transmitted packets increases past the buffer limit, the saturation has little effect on the bandwidth, which averages 1565Mbps. We can conclude that CusComNet is able to process packets at least equal to the rate at which the user application can produce them, even when supplied with a continuous stream of data.

The average latency per packet transmission is found to be $0.83\mu s$ for communication between two directly connected nodes. This varies slightly due to the differences

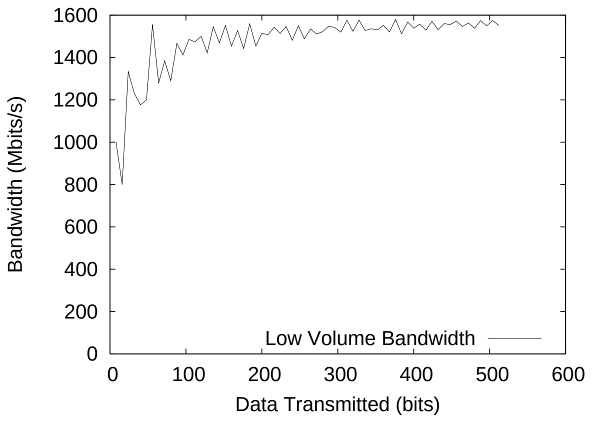

Figure 5. CusComNet's bandwidth for low volume communications.

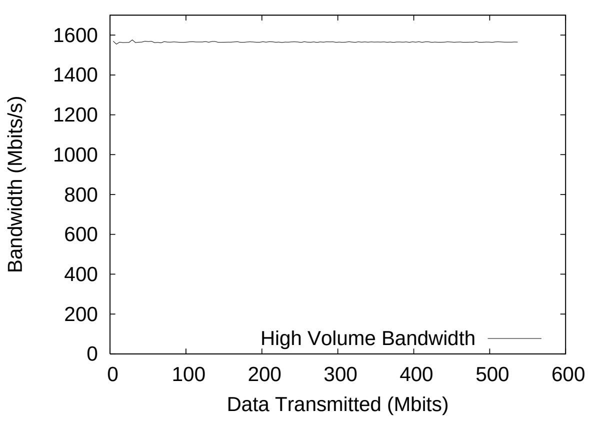

Figure 6. CusComNet's bandwidth for high volume communications.

in the lengths of the cables used to connect each node-pair, but the difference within the prototype design never exceeds $\pm 0.01\mu s$. When increasing the number of packets per transmission, the latency increases uniformly with the number of packets for all communicating node-pairs. When transmitting data via one or more intermediate nodes, the bandwidth is the same as that of directly connected nodes, while each hop increases the latency by $0.57\mu s$.

When dealing with small volumes of data, CusComNet's average bandwidth of 1479Mbps results in an efficiency of between $88-93\%$, given the theoretical line rate of 1600Mbps when using 8B/10B transmission. The uncertainty in the efficiency arises from the reference clock's $\pm 5\%$ error rate. As we increase the packet payload to its maximum value, we find the resulting best-case efficiency rises to between $93-97\%$. For larger scale transmissions, the average bandwidth of 1565Mbps also gives an efficiency of $93-97\%$, however tests across different nodes suggest this value to be closer the 97% limit for a 64 byte CusComNet packet. The remaining inefficiency is caused in part by the packet header, but is primarily due to the transmission protocol used by the data link layer. This protocol is dependent on the connections between each node-pair and is dominated by the inter-FPGA synchronisation: there is at least one synchronisation character—but may require more, depending on the stability of the connection—and synchronisation is performed at regular intervals. Future work will focus on minimising this communication overhead to improve the bandwidth and overall efficiency of the inter-FPGA network.

From the above measurements, we have $B = 1565Mbps$, $T_l = 0\mu s$, $T_r = 0.57\mu s$ and $T_a = 0.13\mu s$. We can use these parameters to estimate the transmission time as shown in Equation 1. For a path including two links (i.e. $k = 2$), a 64-byte packet will take $T = (2-1) \times (0 + 0.57 + 512/1565) + (2 \times 0.13) = 1.16\mu s$. This agrees with the measured results in our experiments. Based on this, we

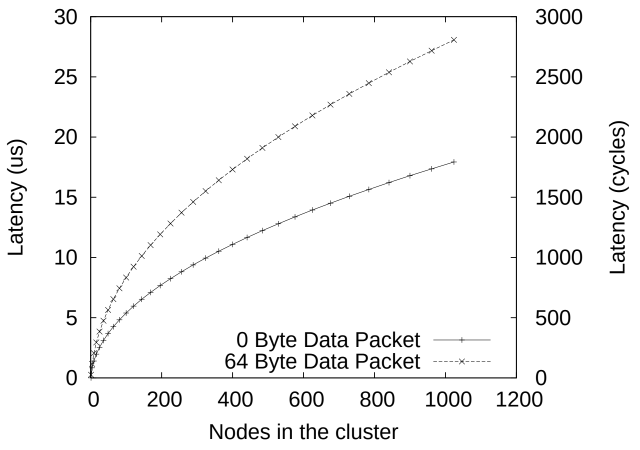

Figure 7. The projected latency when using CusComNet within 2D torus connected clusters.

can project the expected latency value for communications within larger clusters utilising the CusComNet framework. Assuming the use of a 2D torus connection, Figure 7 shows the expected latency when communicating along the longest path in a network, as well as the time taken to transmit a 64 byte data packet. The figure defines the latency both in terms of time for the prototype implementation, and in terms of compute cycles for an application-independent characterisation. Latency is shown for clusters with up to 1024 nodes, but can easily be extended using the above formula to estimate CusComNet's performance for larger clusters.

B. Scalability: N-Body Simulation

The N-Body simulation algorithm is used to simulate the interaction between N particles. This is an iterative process where the position and velocity vectors of a particle are updated by the previous velocity vector and the computed acceleration vector in each iteration. In this work, we implemented the N-Body application using the FPGAs

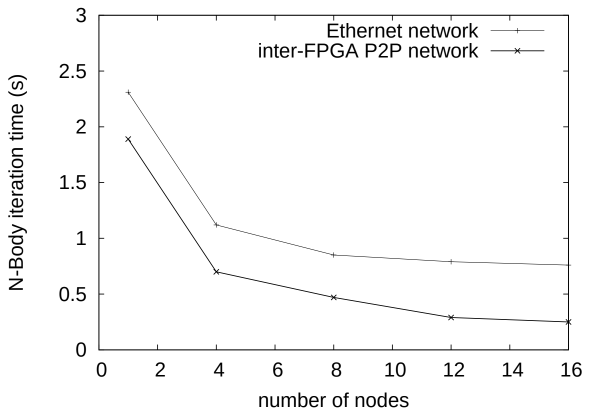

Figure 8. N-Body application scalability using the inter-FPGA network.

as processing units. All FPGA accelerators in the cluster are programmed with the same configuration, while each computes a pre-assigned portion of the total particle space.

Before starting a new iteration, all nodes must have the updated vectors of all the particles. This means that the local partial results in each FPGA must be transferred to all other FPGAs. In an Ethernet only implementation, the local results are distributed through the MPI framework running on the host CPU. With CusComNet, additional circuitry is used to send and record the partial results bypassing the host system. In this case, the iteration management and vector updating are all performed within the FPGA.

In the experiments, the FPGA designs are captured by VHDL descriptions and implemented using the Xilinx ISE 12.3 tool chain. OpenMPI version 1.3.3 is used for inter-CPU communication. The experiments are based on the 4×4 2D torus inter-FPGA network in Figure 2, using the default values for the parameters listed in Table I for FPGA designs. The input data set includes 81920 particles.

Figure 8 shows the improvement gained by migrating the partial result transmission from Ethernet to CusComNet's dedicated inter-FPGA network. The iteration time is obtained by averaging the time of a single iteration without the overhead of FPGA configuration and data initialisation. The results indicate that not only can a 3 times speed-up be achieved for 16 nodes in overall performance, but the performance also continues to improve beyond 8 nodes when using the dedicated inter-FPGA network.

Without a dedicated broadcasting mechanism, the current design has to send the same local results 15 times separately to the 15 nodes. This overhead can be reduced with a broadcast protocol. Based on this implementation, further optimisations can be obtained to allow better application performance. It is possible to overlap the computation and communication times by sending results over the network as soon as they are generated. For a CPU-centric communication system, the overhead of constantly monitoring the FPGA partial results is prohibitively large for this asynchronous communication scheme, but this overhead is much lower for FPGAs.

VII. Conclusion

In this paper we describe CusComNet, a customisable communication network for FPGAs in a computing cluster. The CusComNet framework provides an efficient and versatile data transmission framework with a standardised, easy to use wrapper. CusComNet is implemented in a heterogeneous computing cluster including commodity Gigabit Ethernet and customisable inter-accelerator serial connections. The measured results show that this inter-accelerator network achieves communication throughput of between $88 - 93\%$ of the theoretical limit when transmitting small volumes of data, and between $93 - 97\%$ efficiency for larger data volumes. Communication latency is found to scale linearly as the number of intermediate hops increases when transmitting a packet, and agrees with our latency projections based on a performance model.

Future work will focus on improving the latency and efficiency via low level automated optimisation of the inter-FPGA connections, and the development of adaptive routing algorithms based on the run-time status of the network. High level user interfaces will also be created to ease application development, such as MPI based facilities [8] with automated code generation.

Acknowledgement
This work was supported in part by the UK EPSRC, the European Union Seventh Framework Programme under Grant agreement number 248976 and 257906, and by the HiPEAC NoE, Alpha Data, Maxeler, and Xilinx.

References

[1] *Cray XR1 Reconfigurable Processing Blade*, Cray Inc., 2007.

[2] *SGI RASC RC100 Blade*, Silicon Graphics Inc., 2006.

[3] R. Baxter *et al.*, "Maxwell - a 64 FPGA supercomputer," in *Proc. Conference on Adaptive Hardware and Systems (AHS)*, 2007, pp. 287–294.

[4] R. Sass *et al.*, "Reconfigurable computing cluster (RCC) project: Investigating the feasibility of FPGA-based petascale computing," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2007, pp. 127–140.

[5] *Byte oriented DC balanced (0,4) 8B/10B partitioned block transmission code*, IBM, 1984.

[6] H. C. Tijms, *A First Course in Stochastic Models*. Chichester: Wiley, 2003.

[7] *OMNeT++ User Manual*, OMNeT++, 2010. [Online]. Available: http://www.omnetpp.org/doc/omnetpp41/manual.pdf

[8] M. Saldana *et al.*, "MPI as a programming model for high-performance reconfigurable computers," *ACM Trans. Reconfigurable Technology and System (RTES)*, vol. 3, pp. 22:1–22:29, November 2010.