VELO: A Novel Communication Engine for Ultra-low Latency Message Transfers

Heiner Litz Holger Froening Mondrian Nuessle Ulrich Bruening

University of Heidelberg
Computer Architecture Group
Germany
{heiner.litz, holger.froening, mondrian.nuessle, ulrich.bruening}@ziti.uni-heidelberg.de

ABSTRACT

This paper presents a novel stateless, virtualized communication engine for sub-microsecond latency. Using a Field-Programmable-Gate-Array (FPGA) based prototype we show a latency of 970 ns between two machines with our Virtualized Engine for Low Overhead (VELO). The FPGA device is directly connected to the CPUs by a HyperTransport link. The described hardware architecture is optimized for small messages and avoids the overhead typically found with Direct-Memory Access (DMA) controlled transfers. The stateless approach allows to use the hardware unit directly from many threads and processes simultaneously. It provides a secure user level communication with an extremely optimized start-up phase. Microbenchmarks results are reported both based on proprietary API and OpenMPI basis.

Keywords: fine-grain communication, low-latency message passing, device virtualization, interconnection networks, high-performance computing

1. Introduction

Cluster interconnects typically reach their highest bandwidth at message sizes of 4KB and higher. Such long message transfers are usually executed using DMA in order to offload the CPU. The required initialization of the DMA engine results in an overhead for message start-up. This overhead can be accepted in relation to the long message transfer time.

Short messages suffer from this overhead and therefore *Programmed-I/O (PIO)* is used in this case. Many proposals and implementations have shown that this method can provide low latency and sufficient bandwidth for small messages [1][2][3][4]. Nonetheless, the communication latency is much higher than a memory access to a remote memory module in a NUMA architecture. This is due to a missing communication instruction and the overhead in the protocol conversion from the CPU interface to the network protocol and vice versa.

The primary goal of our design is to close the gap between memory accesses and network transfers. Then the overhead for a small data transfer is low enough to efficiently use fine-grain communication techniques. Data structures do not have to be collected in large bulk transfers, each element can be sent out independently. Typical examples for applications which demand support for fine-grain communication are distributed databases like MySQL Cluster [7] or applications based on the Partitioned Global Address Space (PGAS) model, for instance based on UPC [6]. Such low-latency fine-grain networking will improve scalability of existing applications as well as enable new implementations of communication-bound problems in the many-core era.

The approach to reach this goal is based on a set of techniques which are novel in this combination. These techniques include in particular:

- Stateless work processing
- Secure and atomic triggering from user space
- Integrated end-to-end flow control
- Virtual cut through in all pipeline stages
- Context minimization for virtualization

In particular the atomic triggering and the integrated flow control rely on a seamless integration of the communication engine into the existing system. In [14] it is shown that a highly efficient host interface is necessary to achieve lowest latency. The HyperTransport (HT) technology [8] offers a direct connection from peripheral device to the host CPU, avoiding protocol conversions and intermediate bridges. Its lean protocol is designed for high bandwidth and low latency.

The prototype implementation shown in figure 1 currently uses an 8-bit wide HT200 core [20] providing 800 MB/s of aggregate bi-directional bandwidth and runs internally with 100 MHz clock frequency. To maintain the advantage of several virtual channels and to support multiple functional units in the device, the HT-Core is connected to the VELO unit by means of a crossbar.

The actual process of assembling and retrieving messages is carried out by the VELO *Functional Unit (FU)*. The requester unit is responsible for sending, the completer performs the task of receiving messages and transferring them into main memory, accordingly. The VELO engine is optimized for small messages with low latency, hence applications which have to transmit large messages should employ the Remote Memory Access (RMA) engine to keep CPU utilization low. On every send the API decides whether a message is processed by the VELO or RMA engine. A detailed discussion of the RMA and other possible functional units is out of the scope of this paper. The complete system runs at 100 MHz clock frequency and the links provide a bandwidth of 2 Gbit/s.

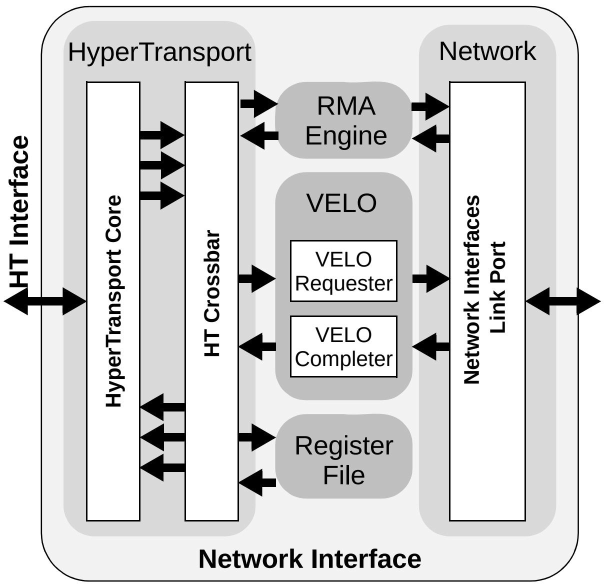

Figure 1. NIC architecture with VELO

The remainder of this paper is structured as follows: In the next section we present a summary of related work in the field of fine grain communication. The properties of the architecture are explained in section 3, while in section 4 details about the implementation and a hardware resource analysis are given. The implementation in hardware allows us to present real-world measurements in section 5, which are enriched by performance results from simulation. In the last section we conclude and provide a brief overview of the future work.

2. Related Work

There have been several attempts with the goal of minimizing communication latency. They can be summed up into well documented research projects which are mostly software oriented and hardware implementations realized by several *High Performance Computing (HPC)* vendors. Some examples for the first group are U-Net [9] or Active Messages [5] developed at the University of Berkeley by Culler and Eicken. Those research groups have been working on software stack optimizations, operating system bypass or latency hiding techniques, however with the disadvantage of not being able to modify the underlying hardware according to their needs. An exception thereof is the DIMMnet project [4] by the Tokyo University which is a hardware-software co-design. It introduced a new low latency technique called AOTF [10] and led to an ASIC implementation which can be directly connected to the CPUs via the system's memory interface. This interface provides much better latency performance than PCI/PCI-X, however it comes with several disadvantages like a restricted unidirectional communication. Other hardware implementations are Quadrics STEN [2] which is a dedicated functional unit for low latency communication which takes advantage of a large SRAM based message buffer. Cray offers communication latency in the range of 2 μs with its RapidArray fabric [11] in the XD1 machines and the SeaStar interconnect [12] in the XT3 machines. Both implementations of Cray are HyperTransport based which seems to emerge as the new standard for low latency communication. Fröning et al. propose another mechanism called ULTRA [13] based on PCI-X which provides latency below 2 μs. One other solution is the InfiniPath [1] adapter from QLogic. It is a streamlined design which lacks many common features like message offloading and RDMA support, however it offers the currently best latency performance of about 1.1 μs. The goal for next generation network interfaces will therefore be to provide sub-microsecond latency. A possible approach is shown in this paper.

3. Architecture

To meet the design goal several issues have to be considered. They include an efficient host interface with an optimized access scheme, a simple conversion from host interface protocol to network protocol (and vice versa) and support for virtualization to allow an unconstrained usage of resources.

Achieving the best network interface performance requires optimization of all system layers. In the case of a network interconnection device this includes the software layer, the system interconnect, the network, link-, and the physical layer of both the sending and receiving side. Furthermore the intermediate switching fabric has to be taken into account.

User-level Communication. From an architectural point of view VELO reuses many common and well researched operation principles like *User-level Communication* [15] and one-copy messaging. It combines them uniquely and enriched with new features, though. User-level Communication to access the NIC directly without O/S involvement for send/receive operations is mandatory for achieving good latency and bandwidth performance. However it removes the O/S's capability of sharing the device among several processes. Systems which implement this technique like [1] and [3] generally restrict the number of concurrent threads which may access the NIC from user space directly to a very limited number. VELO on the other hand offers sending capability to a virtually infinite number of threads. Therefore every thread obtains a mapping of the VELO I/O space into its own address space. This enables a thread to access the NIC by simply writing to the correct address. Security issues and race conditions which generally arise when a single physical resource is shared by many are resolved by introducing virtualization.

Context minimization for virtualization. VELO is a self virtualizing device which allows it to be utilized by a large number of software threads concurrently. In contrast to the concept described by Raj and Schwan in [16], which is based on eight replicated Ethernet communication cores accessed by guest O/Ss in a Virtual Machine environment, VELO allows direct access from user space.

The concept of providing hardware support for virtualization is powerful. However, in general large hardware structures and memories are required to store the context information. To alleviate this problem we propose the novel concept of context minimization for virtualization. This allows user-level communication for up to 64K threads without any O/S intervention.

The basic idea behind this principle is the following. First of all the required context information is reduced to an absolute minimum. In the case of VELO we found out that a secure and reliable communication only requires three attributes to sufficiently describe a message. These are the target node, the receiving thread running on that node and the length of the message. The goal now is to provide this context information together with the data payload to the NIC as efficient as possible. This can be done by including it in the message itself avoiding any additional look-ups. Instead of putting the context in the data payload, VELO defines a new mechanism which reuses the address that is used to access the NIC. Figure 2 shows how to embed context information in the address directly. During system boot the VELO hardware acquires a large *base address register (BAR)* space from the O/S. As 40 or more bit addresses are common in current computing systems this is absolutely tolerable. The key idea now is to use this large address space to encode context information instead of using it in the traditional way of accessing certain registers or memory areas on the device. This allows to easily switch contexts on every reception of a new HyperTransport packet. The parameters *t* (threads) and *n* (nodes) which define the maximum number of nodes and threads in the system can be set arbitrarily. According to the formulas

$$\textit{Number of nodes}\ \ N(n,t) = 2^{(n-t)}$$

and

$$\textit{Number of threads}\ T(n,t) = N(n,t)\cdot 2^{(t-12)}$$

the values n=28 and t=20 for example allow to support 256 nodes with 65,536 threads total, which results in a BAR size of 256 MB.

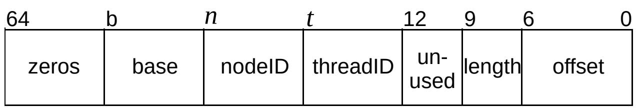

Figure 2. Embedded context information

Another advantage of this approach is the feasible enforcement of access permission. To restrict and manage the access rights of users it is beneficial to use the higher bits in the address for encoding target node and thread. As page mapping is done on a 4 KB basis the upper bits are transparent to the user. This allows the driver to hand out specific communication channels between threads by mapping the corresponding addresses into user space. The message length however is only known by the user (or API) and therefore has to be encoded into the user accessible part of the address. This is represented by the length field in the lower 12 bits of the address. As messages are aligned to cache lines the offset field starts at 0 and is incremented during the transfer.

Atomic PIO message triggering. The scheme introduced above allows the software to send a message with a single HyperTransport transaction and is therefore called atomic. Messages are transferred from the software layer to the NIC using a single PIO access. A single store instruction is sufficient to send a message. It is obvious that this approach provides the optimal latency performance; however it introduces significant overhead which can result in reduced bandwidth. The overhead is caused by the HyperTransport protocol which frames each data packet with a 64 bit command. In the case of a 64 bit data transfer this accords to an overhead of 50%. VELO evades this problem by the aggressive use of write combining. Write combining is a technique supported on x86 systems which allows for bursty transfers when using PIO. This feature which is usually used by graphic adapters can be adopted to VELO by modifying the Memory Type Range Registers (MTRR) which leads to a significant increase in bandwidth. By combining several stores in a single HyperTransport transaction the overhead is reduced to approx. 11.1% in the case of cache line sized messages. Due to the size of the write combining buffers the maximum size of one VELO transaction equals 64 bytes or one cache line.

On the receiver side the software has to poll on a pre-allocated buffer for newly arrived messages which are provided by the NIC through DMA. The buffer is cached by the CPU to avoid unnecessary bus traffic.

Multi-threading support through statelessness. The proposed architecture shows a problem when used in multi threaded environments. Unfortunately thread scheduling may cause the atomicity of messages to be destroyed. VELO messages are generally stored in the write combining buffer and sent out when the maximum length of a cache line is reached to reduce overhead and boost performance. As the x86 architecture does not support writes of a larger size than 128 bit (with special MMX/SSE instructions), write combining of a message may be interrupted if the sending thread is scheduled away during a copy of data larger than 128 bit.[1]

As long as only one thread is using the VELO hardware the message will be eventually continued causing no problems. In a multi threaded environment however messages may interleave each other which leads to the necessity of having a facility which is capable of reassembling messages. One solution would be to provide several queues on the sender side to offer buffering capability for messages of each thread. This however does not scale well as the number of sending threads would be restricted by the number of queues and the buffer utilization would be much worse than compared with a shared queue.

Our preferred solution is to introduce the concept of statelessness which removes any context information, used to describe a thread, from the sender side. By embedding the information of both the original data length and the actual data length into the command packet of the HT transaction the message is implicitly tagged whether it is fragmented. This enables the receiver to merge several parts together to eventually recover the original message. As this process requires a linear search through the receiver memory buffers until all parts are found and as it occurs rarely we decided to carry it out in software. The receive method is non-blocking which means that other messages arriving in the mean time can still be processed. To avoid segmentation of the receive buffer with partitioned messages they are moved in a special recombining buffer. This approach combines high performance and efficiency with a guaranteed data consistency. The marginal penalty of having to recombine messages can be neglected as this occurs very rarely.

1. With a message rate of 3 million messages per second as shown in the evaluation paragraph and a thread scheduler that switches threads every 10ms, one of 30.000 messages will be segmented on average.

Buffer requirements and flow control. To provide a back-to-back stream of messages and to alleviate the impact of network congestions, buffering capability is absolutely mandatory. Determining the correct amount of buffer space can be difficult. Larger buffers usually improve performance, but are more costly in terms of hardware. It is worth mentioning that on-chip memory resources have a cost increase of a factor of 1,000 compared to main memory. This implies that main memory should be used wherever possible and that on-chip memory has to be used in an optimal way because of its size restrictions. To provide decoupling between network and host system VELO uses an on-chip buffer of minimal size. As sending capability to a virtually unlimited number of threads is provided this buffer has to be shared. Virtualizing a single physical hardware resource requires synchronization among the producers. Checking for buffer space negatively affects latency and wastes bandwidth. Even worse, when performed by several producers concurrently, race conditions appear. VELO solves these problems by combining the above principles of atomicity and statelessness with the hardware flow control mechanism of the HyperTransport protocol. HyperTransport provides a credit based flow control mechanism between the peripheral HTX device and the processor which allows optimistic message injection. The need for checking buffer space is superseded while guaranteeing strict data integrity. Discarding of messages can be avoided even with full buffers by generating back pressure to the sender which may then suppress the injection of new messages. Backward flow control is a unique feature of HyperTransport enabling highly efficient injection of messages without the need of checking for queue space.

The flow control of HyperTransport and the network layer is combined to provide a true end-to-end hardware-based flow control. A possible disadvantage of this approach is that congestion in the network may now propagate into the nodes stalling CPU cores. Congestion management techniques [17][18] can diminish this effect. Another solution is to supervise mailbox fill levels from kernel level and hence ensure free buffer space.

In order to achieve lowest latencies virtual cut-through techniques are applied to all stages including buffers.

Message transfer overview.

A message transfer consists of several steps which are shown in figure 3. The sequence starts with the sending thread performing a PIO write onto the requester BAR space. The used address defines the destination and packet length. The write cycle passes the packet payload to the requester using write combining and also triggers the message injection. The requester assembles the packet by preceding the packet with routing and header. The packet traverses the network stages to the target node and is there received by the completer. The completer forwards the packet's header and payload to the receiving thread. In more detail, the packet is stored in a pinned memory region within the address space of the receiving thread. The receiving thread can poll on this buffer and the cache coherency protocol prevents unnecessary main memory accesses.

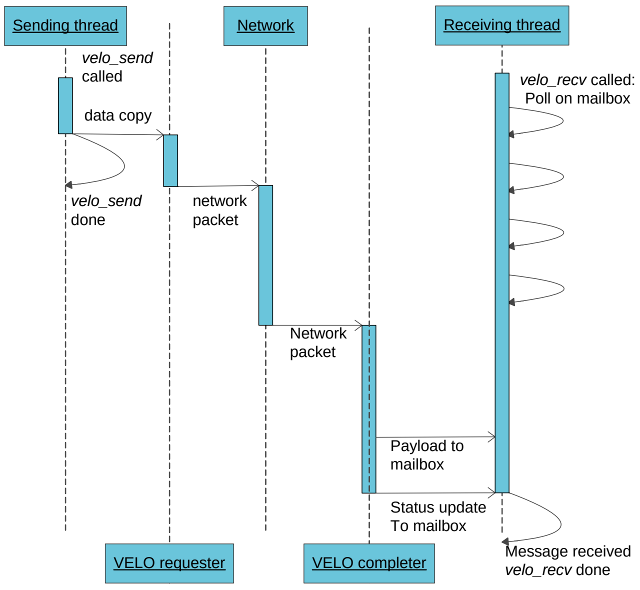

Figure 3. VELO message sequence

4. Implementation

To demonstrate the feasibility of our approach we opted to develop a full-blown FPGA prototype implementation instead of just simulating the design. This requires major efforts however it offers several advantages. First of all results are much more realistic as a prototypes can be used in a real world system controlled by an O/S running several applications. Currently simulators are not even close to emulate a full system including O/S and software in a hardware cycle based fashion. Additionally the simulation time grows dramatically when using more realistic engines. Our prototype is hence millions of times faster than a simulation. With the use of an internal Logic State Analyzer (LSA) e.g. Xilinx Chipscope, detailed and clock cycle accurate performance measurements can be made to evaluate the design. Another benefit is that the prototype can be easily ported into a later ASIC implementation.

The employed testbed consists of a HTX prototyping board [19]. It is equipped with a Xilinx Virtex4 FX-60 FPGA and is perfectly suited for networking applications.

The prototype has been fully implemented in synthesizable RTL level verilog code. It has been aggressively pipelined to support high clock frequencies even on an FPGA without losing focus on the primary concern of building an ultra low latency design. The last building blocks form the lower level network and link layers.

In principle VELO can be build upon any reliable network layer like Infiniband (IB). In this case we have used our own proprietary network interface layer. This sophisticated network layer provides good latency by cut-through forwarding and source-path routing. Other interesting features are virtual channel support, credit based flow control, congestion management, automatic CRC checks and a hardware based automatic re-transmission mechanism. The FPGA prototype implementation used here runs at 100 MHz with a 16 bit wide data-path providing 200 MB/s of raw link bandwidth.

Table 1: VELO resource usage

Component	Usage	percent of FX60
VELO Requester	88 flip-flops 143 LUTs	< 1 %
VELO Completer	274 flip-flops 365 LUTs	< 1 %
VELO Register file	222 flip-flops 199 LUTs	< 1 %

To assess the complexity of VELO, Table 1 summarizes the resource usage of VELO when implementing the Verilog HDL code on a Xilinx FPGA. As the third column shows less than 1% of an Virtex4 FX60 FPGA are used by VELO and the register file portion of VELO (the register file is used by all components of the design). The complete system including HyperTransport IP core, crossbar, network-, link- and physical layer uses less than 60% of the FPGA. we have not specifically performed an ASIC synthesis flow to achieve exact area statistics, but from the FPGA numbers it is clear that ASIC area consumption would be very low (based on flip-flop count). These resource requirements show that VELO is a very compact design in terms of silicon resources.

5. Results

For performance results, both simulation results and measurements from the FPGA based prototype are presented here. The simulation of the complete system is performed using Cadence NCSim and the HyperTransport bus functional models available from the HyperTransport Consortium. Basic latency characteristics of the hardware can be gained from simulation.

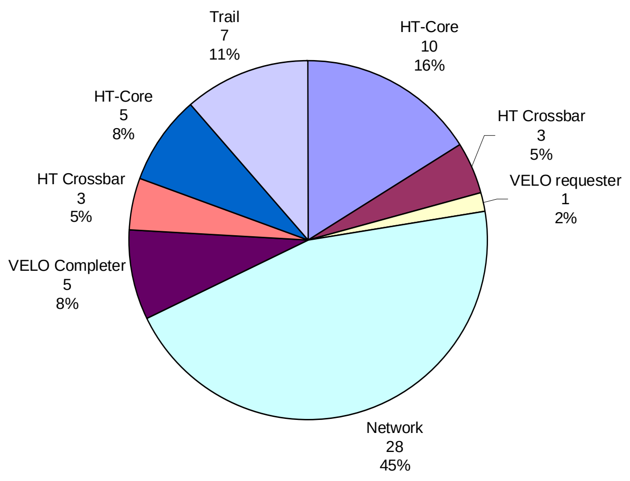

Figure 4. Hardware latencies in cycles from simulation

From the start of sending a HyperTransport packet on the HT link to the transmission of the last word on the HT link at the receiving side, a latency of 64 clock cycles is seen. Figure 4 shows the percentile distribution of the latency to the contributing hardware modules. The VELO requester contributes only 3 clock cycles and the completer 5 clock cycles to the total hardware latency. This shows that the VELO mechanism itself is highly efficient considering latency. The HyperTransport core and the HT crossbar contribute 21 cycles. Traversal of the network from node to node finally adds 28 cycles.

From the clock cycle count, latency times can be derived. In the FPGA implementation with its 100 MHz cycle time, this translates to a hardware latency of 640 ns. In contrast an ASIC implementation would easily be able to reach a 500 MHz clock frequency thus pushing latency down to a mere 130 ns, which is in the order of a main memory access.

In a real system latency is increased by hardware within the CPU (northbridge) and the memory controller, as well as by the instructions the software layers need to execute in order to send and receive messages. To quantify this latency as well as gain real-world measurements, the complete system is loaded onto two HTX Boards. Measurements are conducted on two identical machines featuring each an IWill DK8-HTX motherboard, one dual-core Opteron 870HE and 2GB of RAM. OpenSUSE 10.0 is used as operating system. For comparison the tests are also conducted with Infinipath HTX HCAs [1].

To use VELO from application software, driver software and API middleware has to be provided. For this purpose a Linux kernel driver is developed to manage allocation of resources to client processes, e.g. allocation of receiver thread IDs together with the associated memory blocks. Also, access control, configuration and network control is handled by the kernel-level driver. The second software component is the VELO API which abstracts both interaction with the kernel level driver (to allocate resource etc.) as well as direct user-space interaction with the hardware itself. For parallel programming the de-facto standard is MPI [21]. To support MPI, a bit-transfer-layer (BTL) component for OpenMPI [22] is developed which is layered on top of the VELO API.

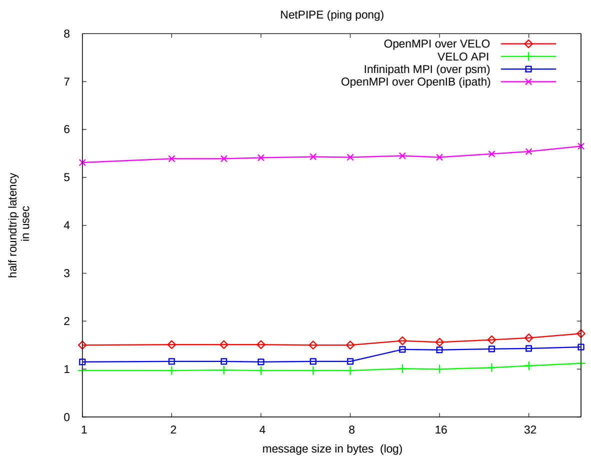

Figure 5. VELO half-round-trip latencies (NetPIPE)

To put the results into scope, the Qlogic Infinipath HTX network is benchmarked in the same system. This NIC is implemented in an ASIC also connected to the host via a 16-bit HT800 HyperTransport link. On the network side the controller uses a 4x single-data-rate Infiniband interface. The InfiniPath ASIC provides eight times the bandwidth and clock rate at the HyperTransport interface and five times the bandwidth on the network side compared to the VELO FPGA implementation described above. The InfiniPath Software version 1.2 together with OpenIB is used to test MPI performance, both using the InfiniPath MPI and OpenMPI over OpenIB. InfiniPath MPI is a specially optimized MPI version which implements a thin layer on top of the hardware. The OpenIB version seems to incur a much higher overhead, probably since standard IB interfaces have to implemented by software. The two InfiniPath HTX HCAs were directly connected, so no switching delays are encountered.

As a micro benchmark, NetPIPE [23] was chosen. The code was adapted to enable NetPIPE to run directly on top of the VELO API. The benchmark was performed for small messages, the area of application for VELO. So, the maximum message size is chosen to be smaller than one cache line to facilitate a one-to-one mapping from higher-level messages to VELO transactions. Figure 5 shows the latencies of messages from 1 to 48 byte payloads using the API and the different MPI versions. For VELO, the result for the half-round-trip latency for minimum sized messages is 0.97 μs. This latency increases to 1.50 μs when using OpenMPI.

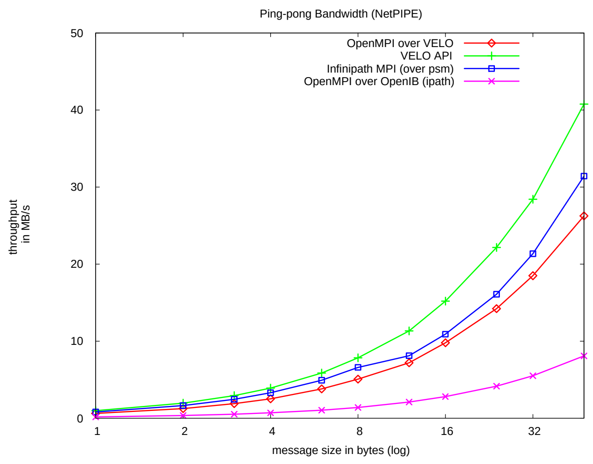

Figure 6. Ping-pong bandwidth (NetPIPE)

CPU, memory controller and software overhead add about 350 ns to the pure hardware latency seen in simulations. Adding MPI on top of this adds another ~500 ns. This is probably due to the layered component model of OpenMPI that generally eases implementation of a new interconnection network, but also, in the case that one only writes the lowest layer (the BTL), less than optimal performance.

In comparison, the Infinipath HTX adapter, well known for its especially low latency, reaches 1.14μs using Infinipath MPI and 5.31μs latency over OpenMPI/OpenIB. As expected the Infinipath MPI performs similar to a propriatary API implementation, while the more generic OpenMPI/OpenIB with it's additional software layers incurs a latency penalty. So the Infinipath MPI implementation performs similar to our direct API.

When compared to one of the best ASIC based interconnection solutions currently available, the VELO architecture provides exceptional low latency for small messages.

Figure 6 shows the ping-pong bandwidths. Again VELO performs excellent, with a bandwidth of 7.8 MB/s at 8 byte message size compared to the 6.5 MB/s of Infinipath HTX. Finally, figure 7 shows the NetPIPE streaming bandwidth available to messages of size 1 to 32, both using VELO and InfiniPath HTX. VELO achieves a very high bandwidth of nearly 100 MB/s for 32 byte messages, Infinipath HTX reaches about 67 MB/s. The N½ message size of VELO is as low as 32 byte since 50% of the theoretical peak link bandwidth is reached at this message size.

Often the message rate is referred, too. The maximum message rate for VELO is 4.8 million messages/s using one core; InfiniPath HTX reaches 3.0 million message/s in the tested system.When sending 32-byte sized message the rate is still over 3 million messages/s.

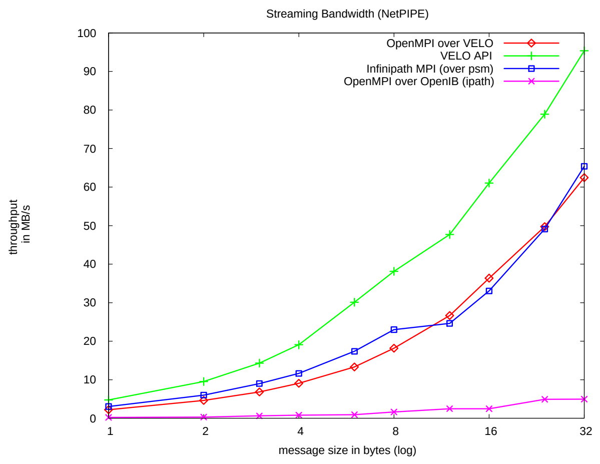

Figure 7. Streaming bandwidth (NetPIPE)

6. Conclusion and Outlook

We have shown that the highly efficient VELO functional unit in combination with the HyperTransport interface provides an outstanding low latency of 970 ns between two nodes, in spite of an FPGA implementation with a mere clock frequency of 100 MHz. This particularly demonstrates the excellent performance of the chosen architecture. To the best of our knowledge this is the best latency ever reached on standard computing hardware with FPGA based NICs. Even in comparison with commercially available, ASIC-based high-performance networks these results are very promising.

The initial MPI performance numbers are very encouraging both for the VELO hardware as well as the software environment. The architecture principle of stateless, virtualized functional units allows very low latency communication for short messages.

More complex application-based tests will be carried out once a larger system becomes available. Additionally, the benefit of a complete self-virtualized communication device will be shown when a large number of user processes are simultaneously accessing the device. Not only virtual machine environments, but also multi-core systems can benefit a lot from shifting the virtualization overhead from software layers to hardware.

Finally, there are still possibilities to further enhance the performance. In the future an ASIC implementation will show a significant decrease of latency and a much higher bandwidth. In particular moving to an ASIC technology will scale the hardware latency linear with the clock frequency. Keeping the CPU and software latency components at the same level of 300 ns a half-round-trip latency of less than 0.5 μ s is feasible.

7. References

[1] L. Dickman, G. Lindahl, D. Olson, J. Rubin, J. Broughton. PathScale InfiniPath: A First Look, *Proc. of the 13th Symposium on High Performance Interconnects*, Washington, DC, 2005.

[2] F. Petrini, W. Feng, A. Hoisie, S. Coll, E. Frachtenberg. The Quadrics Network: High-Performance Clustering Technology, *IEEE Micro, 22(1):46-57*, 2002.

[3] H. Fröning, M. Nüssle, D. Slogsnat, P. R. Haspel, U. Brüning. Performance Evaluation of the ATOLL Interconnect, *Proc. of IASTED Conf. on Parallel and Distributed Computing and Networks (PDCN),* Innsbruck, Austria, 2005.

[4] Konosuke Watanabe, et.al. Martini: A network interface controller chip for high-performance computing with distributed PCs, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 18(9), Sept. 2007.

[5] T. von Eicken, D. E. Culler, S. C. Goldstein, K. E. Schauser. Active messages: a mechanism for integrated communication and computation, *Proc. of the 19th Annual International Symposium on Computer Architecture*, Queensland, Australia, 1992.

[6] W. Chen, C. Iancu, K. Yelick. Communication Optimizations for Fine-Grained UPC Applications, *Proc. of the 14th International Conf. on Parallel Architectures and Compilation Techniques*, Washington, DC, U. S., 2005.

[7] MySQL AB, Benchmarking Highly Scalable MySQL Clusters, Technical White Paper, 2007

[8] Hypertransport Consortium, *HyperTransport Technology I/O Link - White Paper*, www.hypertransport.org, July 2001.

[9] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing, *Proceedings of the fifteenth ACM symposium on Operating systems principles.* 1995

[10] Noboru Tanabe, Junji Yamamoto, Hiroaki Nishi, Tomohiro Kudoh. On-the-fly Sending: A Low Latency High Bandwidth Message TransferMechanism, *2000 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '00),* 2000.

[11] Brightwell, R. Doerfler, D. Underwood, K.D. A preliminary analysis of the InfiniPath and XD1 network interfaces, *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International.*

[12] Ron Brightwell, Kevin Pedretti, Keith D. Underwood. Initial Performance Evaluation of the Cray SeaStar Interconnect, *Proceedings of the 13th Symposium on High Performance Interconnects,* 2006.

[13] H. Fröning, H. Litz, U. Brüning. A new Ultra-low Latency Message Transfer Mechanism, *Proc. of IASTED Conference: Communication Systems and Networks (CSN 2007),* Aug. 29 - 31, 2007, Palma de Mallorca, Spain.

[14] J. Beecroft, D. Addison, F. Petrini, M. McLaren. Quadrics QsNetII: A Network for Supercomputing Applications, *Proc. of Hot Chips 15,* Palo Alto, California, United States, 2003.

[15] E.W. Felten, R.D. Alpert, A. Bilas, M.A. Blumrich, D.W. Clark, S. Damianakis, C. Dubnicki, L. Iftode, K. Li. Early experience with message-passing on the shrimp multicomputer, *Proc. of the 23rd International Symposium on Computer Architecture (ISCA23),* 1996.

[16] Himanshu Raj, Karsten Schwan. High Performance and Scalable I/O Virtualization via Self-Virtualized Devices, *Proc. of IEEE International Symposium on High-Performance Distributed Computing (HPDC),* Monterey, California, USA, 2007.

[17] Jose Duato, Ian Johnson, Jose Flich, Finbar Naven, Pedro Garcia, Teresa Nachiondo. A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks, *Proc. of the 11th international Symposium on High-Performance Computer Architecture (HPCA)*, 2005

[18] Elvira Baydal, Pedro Lopez, Jose Duato. A Family of Mechanisms for Congestion Control in Wormhole Networks, *IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 9, pp. 772-784,* 2005.

[19] Holger Fröning, Mondrian Nüssle, David Slogsnat, Heiner Litz, Ulrich Brüning. The HTX-Board: A Rapid Prototyping Station, *3rd annual FPGAworld Conference, Nov. 16, 2006,* Stockholm, Sweden.

[20] David Slogsnat, Alexander Giese and Ulrich Bruening. A versatile, low latency HyperTransport core, *Fifteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays,* Monterey, California, February 2007.

[21] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, www.mpi-forum.org, 1994.

[22] Edgar Gabriel, et al. *Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation*, EuroPVM/MPI, 2004.

[23] *NetPIPE,* http://www.scl.ameslab.gov/netpipe.