A NEW ULTRA-LOW LATENCY MESSAGE TRANSFER MECHANISM

Holger Fröning, Heiner Litz and Ulrich Brüning
Computer Architecture Group, University of Mannheim, Germany
{froening, heiner.litz, bruening}@uni-mannheim.de

ABSTRACT

Cluster computing is still the most cost-effective solution to meet the increasing demand for computing power. Clusters are typically based on commodity computing hardware with specialized *interconnection networks (IN)*. These cluster interconnects differ from commodity networks by higher bandwidth, lower latency, lower CPU utilization and improved scalability. But even with these sophisticated INs the latency of a message transfer between two nodes is still decades higher than a local memory access. Especially for fine grain communication the latency of a message transfer is crucial. An analysis of the latency shows that the main component originates from the I/O system. The goal of this paper is to present a new mechanism called *Ultra Low Latency Message Transfer (ULTRA)*, which allows message passing with lowest latencies possible. Beside the usage of well-known techniques like User-Level Communication this work focuses on improving the Network Interface by an optimized and most efficient usage of the I/O system. The ULTRA mechanism and architecture presented here show a topmost optimized approach for low latencies, limited only by the used standard I/O system. With it a much closer coupling of the cluster nodes is possible and fine grain communication schemes are more suitable for cluster computing.

KEYWORDS: High Speed Networking, Network Interface Architecture, Low Latency Message Passing, Fine Grain Communication

1 Introduction

The demands for computing power are steadily increasing. Regarding high performance computing clusters are the most cost effective solution. This is also substantiated by the biannual TOP500 list [1], where the 500 fastest supercomputers of the world are enlisted. In the current list (November 2006) 361 cluster based supercomputers are listed, which is more than 70% of the 500 fastest supercomputers. Typically clusters are based on commodity processors with specialized *interconnection networks (IN)*. Special INs are used because standard INs like *Gigabit Ethernet* cannot meet the demands regarding performance (start-up latency and peak bandwidth) and scalability. Such specialized INs are for instance *QsNet2* [2] *by Quadrics, Myrinet* [3] *by Myricom* or *Infiniband* [4]. Beside these widely used INs several custom INs do exist. All of them focus on scalability, high performance regarding latency and bandwidth, and reliability. But even these specialized INs are only loosely coupled over a standard I/O interface to the main processor of the node[1]. Typical standard I/O interfaces are for instance *PCI, PCI-X* [5] or *PCI-Express* [6].

The performance gap between CPU to memory and CPU to I/O limits the performance of INs. A local access to memory is much faster than a remote access (either over message passing or Remote Memory Access), because INs cannot avoid the I/O interface as they rely on a widely available interface. Hence this interface has to be used in a very efficient way. Especially the latency increase with every additional I/O cycle is remarkable.

For a close coupling of the different cluster nodes, the latency gap between local and remote accesses must be as small as possible. Only with such a tight coupling *fine grain communication schemes* are possible. They are based an a large number of small messages, in opposition to coarse-grained schemes with fewer but larger messages. Obviously fine grain communication schemes are only efficient if the overhead per message is small enough.

The remaining of the work is structured as follows. Section 2 motivates the work presented and shortly introduces other low-latency approaches. In section 3 the basic architecture is presented. An evaluation of the architecture is given in section 4 together with simulation results. Section 5 concludes and gives a short outlook about the next steps.

2 Motivation

Goal of ULTRA is the development of a communication technique which reduces the latency of a message transfer between two nodes to it's minimum. This allows a shift from coarse grain to fine grain communication and the system becomes closer coupled. Instead of collecting many small data structures into large bulk messages, these small elements can be sent out independently. *Active messages* [7] (which are sent out to trigger certain operations) are another application where ULTRA fits perfectly. Also

1. For clusters based on custom processors the IN is sometimes directly connected to the CPU. These special cases are not limited by the I/O interfaces.

fine-grained *Global Address Space (GAS)* applications can be improved a lot, rendering software assistance unnecessary [8].

State of the art. There are several other approaches trying to reduce the latency of a message transfer. Basically they differ regarding the location of the *Network Interface Controller (NIC)* and if specialized or standardized interfaces are used. Some systems integrate the communication modules into the main processor, which is certainly the best approach to minimize latency. But the use is restricted to the used processor type. Examples for such systems are the Transputer [9] or the iWarp [10]. This approach requires a new processor design or the modification of an existing one, which is not the goal of this work. Another approach is DimmNet-1 [11], which is based on a network interface plugged into a DIMM socket of the mainboard. These sockets allow much faster accesses than I/O sockets, but the use is very restricted: DIMM devices cannot signal any events to the processor or invalidate cache lines to enforce a re-read. An example for a specialized solution is the RapidArray fabric used in the XD1 system from Cray [12]. It's use is limited to this system. Few details are known, but the network interface embeds processors to off load network functions. Memory copies are used to transfer small messages onto the NIC.

A special solution is not targeted with ULTRA, instead it should be usable in widely available systems by using a standard I/O interface. Examples for such systems are Quadrics STEN unit, InfiniPath by QLogic (formerly PathScale) [13] or the research project ATOLL [14]. While the exact functionality of the commercial solutions is best to our knowledge not published, the ATOLL approach is well documented.

ATOLL provides two mechanism for message transfer, the *Programmed-I/O (PIO)* and the *Direct Memory Access (DMA)* mode [15]. Because of the overhead to initiate a DMA transfer this mode is most suitable for large messages. Small messages (for instance with a size below 512 bytes) are optimally transferred using the PIO mode. PIO means the processor copies each data word into the NIC. The number of I/O cycles required is the main component of the total latency. For ATOLL, several PIO writes to different registers are required. This approach can still be significantly improved by introducing bursts.

The basic idea of ULTRA. If the hardware latency of a message transfer is examined in more detail, it can be seen that the main component originates from the I/O interface [16]. The remaining parts like network device and switch fall-through latencies are much smaller. For instance the fall-through latency of an ATOLL switch is about 90ns [14], while a PCI I/O cycle requires about 500ns.

Focus of the development is the improvement of the NIC architecture. The switching network behind is not modified by ULTRA, a successor of the ATOLL switching units are used here. To allow a broad use of ULTRA a standard I/O interface is inevitable. Therefore the most critical part of ULTRA is a highly efficient use of this interface, otherwise it turns into a bottleneck. The number of required I/O cycles to inject and retrieve messages is the dominant part of the total latency. Minimum latencies are only achievable with one cycle for each message sending and receiving. User-Level Communication [17] is inevitable, otherwise the required system calls would dramatically increase the latency of a message.

The NIC architecture developed here provides a message transfer mechanism with minimal latency. Some restrictions are introduced in order to achieve this. The maximum message size is limited as well as the number of destinations. To not restrict ULTRA to special use cases, the mechanisms developed here are only part of a communication set. Beside ULTRA the IN provides other methods to support bulk transfers with larger payloads. This is comparable to modern CISC architectures with their manifold instruction set, each optimized for a special use. Compilers or (communication) libraries are responsible for choosing the most appropriate instruction or communication method.

3 Basic Architecture

ULTRA is comprised of a sophisticated hardware architecture and a very efficient, low overhead software interface. Both will be presented in this section. Key elements include minimization of the packet initialization and completion overhead by *pre-initialization* and *pre-completion*, a very efficient usage of the I/O interface, an atomic packet injection trigger mechanism and a latency optimized packet format. The main functional units which are responsible for the transfer of ULTRA packets are the *Requester* module which resides on the sender node and the *Completer* module which is located in the receiver node. To support a bidirectional communication scheme every node therefore has to implement both modules.

The application accesses the Requester to send out data. The Requester assembles the packet and forwards it to the network port, which serializes the packet and partitions it into CRC-protected *Flow Controlled Units (FLITs)*. The FLITs are injected into the network. The routing frame preceeding the header and payload of the packet allows it to be forwarded to the correct end node, where it is passed to the Completer unit. The Completer writes the packet into main memory, from where it can be fetched by the receiving application.

3.1 Requester

The Requester shown in figure 1 has the function of providing up to eight independent ULTRA ports which can be simultaneously used by applications to inject packets into the network. Each port is pre-initialized with routing and header informations. These are considered as the static part of a packet. The dynamic part is the payload including the tag. Only this part must be delivered as payload. It is

written in a single burst to the Requester, which also triggers the dispatch of a packet. The time required to construct a packet header and to determine the appropriate routing is completely avoided.

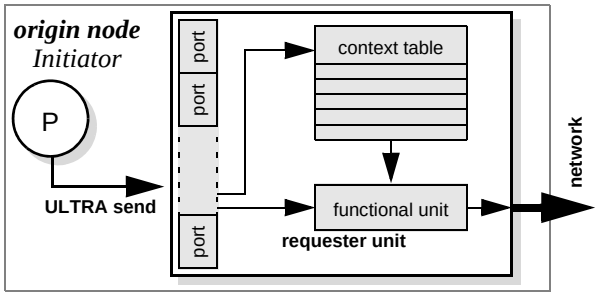

Figure 1: Basic functionality of an ULTRA requester unit

The block diagram of the Requester is shown in figure 2. It consists of a data structure called *Descriptor Look-up Table (DLT)* holding an entry for each ULTRA port. An entry contains the routing, the *Target ULTRA Port Identification (TUPID)* and all header information needed to successfully deliver a packet to the target Completer port. Another major advantage of using packet pre-initialization is the reduced amount of data which has to be transferred from the host to the peripheral device, resulting in a minimum number of required I/O interface accesses.

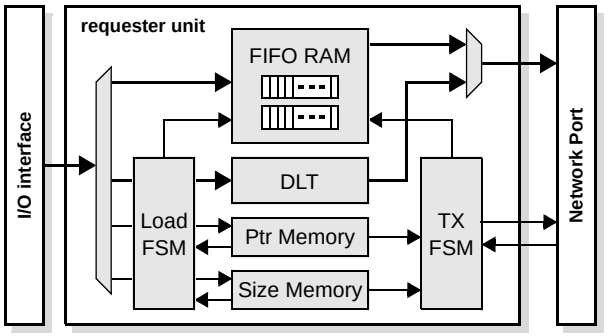

Figure 2: ULTRA Requester block diagram

The payload is appended to the pre-constructed packet by the *Transmit Finite State Machine (TX FSM)* by fetching it from the *FIFO RAM* where it was inserted previously by the *Load FSM*. This data structure is an SRAM organized as multiple FIFO queues of configurable length. In our implementation it is a RAM block of 1KB size. It is managed by the Pointer Memory and holds a variable amount of up to eight queues of configurable size to provide a data buffer for each one of the ULTRA ports. Every port can be assigned an amount of buffer space depending on its demands.

3.2 Completer

Figure 3 shows the architecture of the Completer. It is similar to the Requester's, however, the direction of the data flow is inversed. The *Receive FSM (RX FSM)* interprets the TUPID stored in the header to determine the correct receive buffer. Furthermore, the packet is checked for authorization to prevent flooding of the Completer and for correct format.

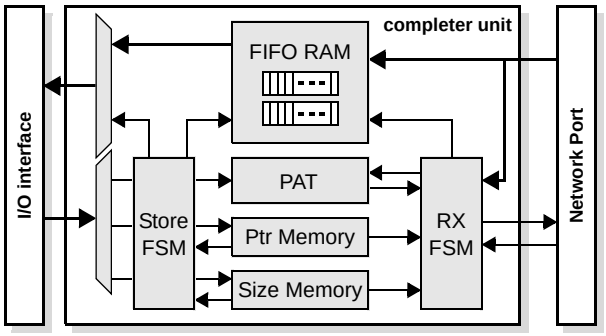

Figure 3: ULTRA Completer block diagram

As ULTRA packets contain no tag which identify or map them to a certain request basically every packet is capable of being consumed by the Completer. This problem is addressed by the introduction of the *Port Authorization Table (PAT)*. For every UPID the PAT holds several entries which store information about authorized communication partners for this particular port. The PAT is managed by the driver in software which creates or deletes entries depending on the currently allocated ULTRA connections between the nodes. Every packet which fails the authorization check is either directly discarded or a system interrupt is thrown.

The other components of the Completer resemble their pendants in the Requester. The register structures *Pointer Memory* and *Size Memory* are responsible to manage the FIFO RAM which buffers the received packets of different UPIDs. The Store FSM writes the payload using DMA back into main memory.

3.3 Packet Injection

In order to minimize the latency of a message passing system it is not sufficient to only provide an efficient hardware platform but also to optimize the software interface as much as possible. User level communication and the avoidance of interrupts are inevitable to achieve high performance. However, every detail has to be considered to attain best results. As described above the optimal utilization of the I/O interface is extremely important. This is only possible if the number of I/O cycles is minimized and burst accesses are used instead of single cycles. Further key components are the use of an efficient synchronization scheme between the user process and the hardware and a low overhead packet format.

The injection of a packet into the network is carried out from user space simply by writing data to a specific address. The transmission of the packet is completely transparent to the user and appears similar to a Two-Register Interface described by Dally in [18]. Both the Requester and the Completer have to be mapped into user space by the device driver. The user is than enabled to

access the network without interaction of the *operating system (O/S)*. To further improve the performance, a sophisticated addressing scheme is used to encapsulate configuration data into a data transfer. The basic idea is to use the 12 lower significant bits of the address to encode additional information. The smallest amount of memory which can be mapped in today's O/S equals the size of a page (4kB), so there is no waste in address space within this approach.

Figure 4: User access address coding

Figure 4 shows the page offset address used to access the Requester. Bits 7 through 9 define whether the packet should throw a notification, interrupt or be acknowledged by the receiver. The most important field is the packet length (bits 3-6). It ranges from one quad word (64 bit) to the size of one cache line (64 byte) plus an additional tag. The lowest three bits cannot be used due to alignment.

For performance reasons it is desired to use bursts whenever possible to transfer data to the device which can be realized by using the following technique. The precondition for bursts is that all data forming a burst access is written to consecutive addresses. This is assured by retaining the upper five bits of the address during a burst and incrementing the packet length instead of decrementing it while writing the data. This implies that a packet length of zero in fact represents a packet length of seven and vice versa.

3.4 Packet Retrieval

On the receiver side an efficient software interface is even more important as the receiving process has no anticipation of the point in time a packet arrives. Therefore either a polling scheme or interrupt mechanism has to be implemented to inform the receiver of a newly arrived packet. Using an interrupt notification has been ruled out as it involves a very time consuming trap into the O/S. Polling can be accomplished by accessing directly the device. But it causes unnecessary I/O traffic, therefore this possibility is discarded. The other possibility is to use a main memory region to store received data. Performance examinations of the PCI bus also showed that a memory write access from device to memory is much faster than I/O accesses [19]. Therefore the Completer writes received packets to main memory from where they can be fetched by the user process. The cache coherency protocol can be used to reduce the number of main memory accesses significantly during polling, independent of invalidate or update policy. To store the data in main memory some requirements must be considered. The utilization of bursts is mandatory for lowest latencies. Furthermore an efficient synchronization technique to manage the receive buffer has to be deployed. On this account the receive queue is partitioned into several *ULTRA Port Slices (UPS)* of 11 x 64 bit as shown in figure 5. Every slice accommodates a whole cache line of data, two 64 bit tags and a status word. The status word is used to store the length and to mark an UPS as valid.

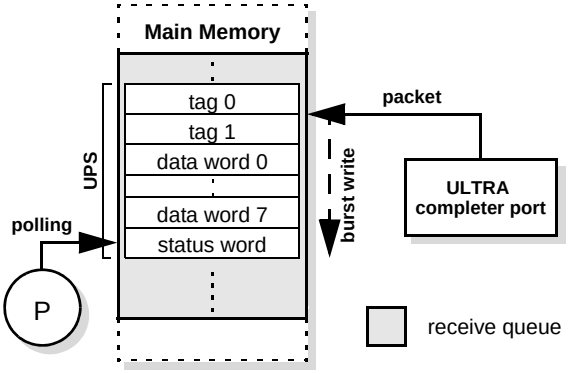

Figure 5: Partitioning of the receive queue

The fields are arranged in a consecutive order to allow writing of an ULTRA port slice by using a single burst access. The key feature of a UPS is that transfers always end at the status word. This enables the Host to poll a defined memory address without any knowledge about the length of the packet. Rather this information is embedded into the last field allowing the host to read out the whole UPS as soon as it is valid independently from it's size. As the Completer updates the valid field after writing the data no race conditions may occur.

4 Evaluation

The ULTRA presented in this work has been fully implemented in synthesizable RTL-level Verilog code. To achieve a better test coverage it is not only simulated but also mapped and tested on an FPGA. This allows for both functional verification and performance evaluation in a real world system. The fully pipelined code is synthesized for a Xilinx Virtex 4 device at 200 MHz. Despite such low clock frequencies ULTRA already shows good results. An ASIC implementation should be easily able to improve performance by a factor of 3 to 5. This should be considered when comparing it to other ASIC based architectures.

As the focus of this paper lies on a specific functional unit used for sending and receiving packets rather than a complete network we concentrate the evaluation on ULTRA itself. To give a better insight figure 6 shows a waveform of the Requester which is extracted from the simulation environment. It shows the complete process of sending a packet including the Load and the Transmit FSM in their successive states. The upper part of the waveform shows some host interface signals and the transfer of the packet into the FIFO RAM. The lower half of the waveform displays the sending process and the link interface signals comprised of the start and end of packet signals and a frame identifier.

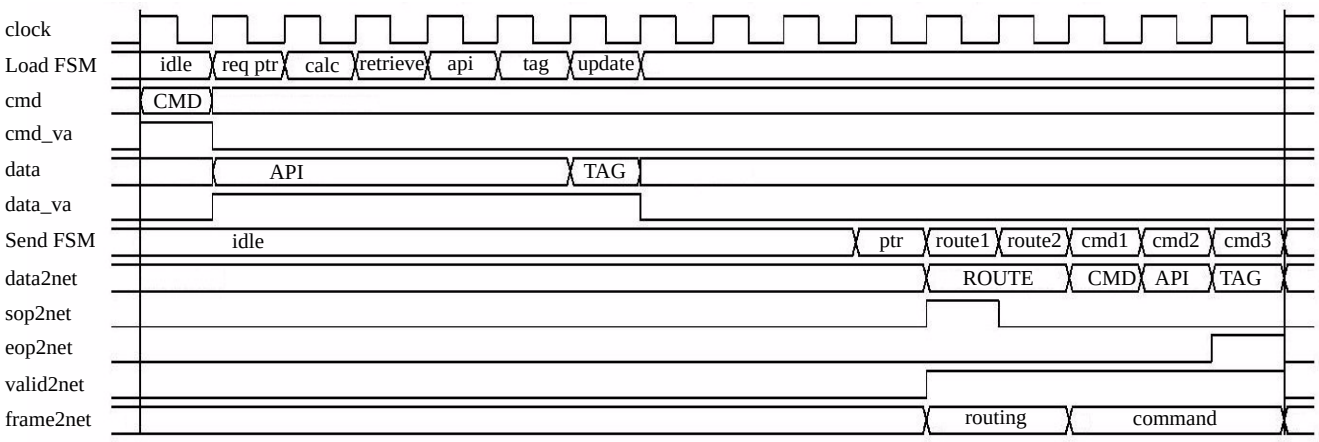

Figure 6: Simulation Waveform of Requester Unit

Simulation results. For the evaluation of latency and bandwidth the Requester and Completer are connected to each other. Different packet sizes are used to show their impact on performance. Note that an ULTRA packet always carries 128 bit of tag information in it's command frame which is mandatory for an efficient MPI implementation. The minimum size of a zero payload packet is therefore 2 quad words and the maximum size is 10.

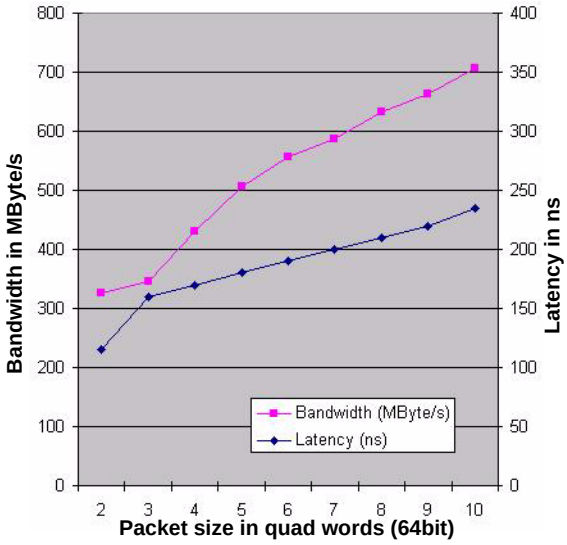

Figure 7: Simulation results

Figure 7 shows the measurements obtained from simulation. For the minimum size of 2 quad words the start-up latency is only 115ns, while already a bandwidth of 325MByte/s is achieved. The increase of latency is low, for the maximum payload (which equals one cache line) only 380ns are required. The bandwidth for this size exceeds 700MByte/s.

To compare an ULTRA enabled network system to other competitors like Infiniband or Myrinet, the latency of the host interface and the network switch have to be taken into account. The host interface protocol which currently provides the best performance regarding latency is *HyperTransport (HT)* [20]. HT is an established high performance standard protocol design for board-to-board and chip-to-chip interconnects. A HT core which is developed in [21] shows latencies of 11 cycles when receiving and 7 cycles when sending data towards the host. The full round trip latency of the core including the software layer is 300ns. Combined with the ATOLL switch presented in [10] which has a switching delay of 27 cycles this architecture is capable of providing an aggregated latency of below 1 us.

5 Conclusion and Outlook

The techniques presented in this work are used to implement a message passing mechanism which provides lowest possible latencies. Standard I/O interface like PCI-Express or HyperTransport can be used, preventing a restriction to special systems. This is achieved by using an extremely efficient packet injection and retrieval method coupled with a sophisticated software interface which causes very low overhead. One single I/O cycle is sufficient to inject a packet into the network. For packet retrieval, the receiving process does not need to access the device at all: it polls on a replicator located in main memory. The cache coherence protocol prevents unnecessary traffic on the system interconnect. Only if changes in the replicator area occur, the updated value is fetched from main memory. The I/O traffic is also minimized.

In order to show ULTRA's performance, the results from simulation runs are shown. Taking into account the limited operation frequency of 200MHz ULTRA shows an excellent hardware latency of only 115ns. Combined with a HyperTransport core and a low latency switch overall latencies below 1000ns are possible. Best to our knowledge this is yet not achieved by INs based on standard I/O

interfaces. In particular it shows the efficiency and performance of the ULTRA approach.

Next steps will be the development of a suitable API to support higher level protocols like OpenMPI. Beside this, a newly developed Rapid Prototyping Station [22] based on an FPGA and a HTX I/O interface [23] will be used as platform. HTX is a connector on a motherboard which enables a direct connection to the CPU interface for add-in cards. The missing intermediate bridges result in a reduced I/O latency. Combining ULTRA and the HT interface will further reduce the latency for message passing systems and provide a new level of communication speed.

6 References

[1] TOP500 Supercomputer Sites, http://www.top500.org.

[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] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, W. Su. Myrinet: A Gigabit-persecond Local Area Network, *IEEE Micro, 15(1):29-36*, February 1995.

[4] Infiniband Trade Association. InfiniBand Architecture Specification Release 1.2, available from http://www.infinibandta.org.

[5] E. Solari, G. Solari, W. Solari: *PCI & PCI-X Hardware and Software: Architecture and Design* (5th Edition, Annabooks, San Diego, 2001).

[6] PCI SIG, PCI Express base specification 1.0a, 2002.

[7] 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.

[8] 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.

[9] C. Whitby-Strevens. The Transputer, *Proc. of the 12th Annual International Symposium on Computer Architecture*, Boston, Massachusetts, United States, 1985.

[10] T. Gross, D. R. O'Hallaron. *IWarp. Anatomy of a Parallel Computing System* (MIT Press, Cambridge, Massachusetts, U. S., 1998).

[11] N. Tanabe, J. Yamamoto, H. Nishi, T. Kudoh, Y. Hamada, H. Nakajo, H. Amano. Low Latency High Bandwidth Message Transfer Mechanisms for a Network Interface Plugged into a Memory Slot, *Cluster Computing 5(1):7-17*, January 2002.

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

[13] 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.

[14] 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.

[15] U. Brüning, L. Schaelicke. Atoll: A High- Performance Communication Device for Parallel Systems, *Proc. of 1997 Conf. on Advances in Parallel and Distributed Computing*, Shanghai, China, 1997.

[16] 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.

[17] 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.

[18] W.J. Dally, B. Towles: *Principles and Practices of Interconnection Networks* (Morgan Kaufman, San Fransisco, 2004).

[19] H. Litz. *Advanced Hardware Communication Techniques*. Diploma thesis presented to the Department of Computer Engineering, University of Mannheim, Germany, 2005.

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

[21] D. Slogsnat, A. Giese, U. Brüning: A versatile, low latency HyperTransport core, *Proc. of 15th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, California, February 2007.

[22] H. Fröning, M. Nüssle, D. Slogsnat, H. Litz, U. Brüning. The HTX-Board: A Rapid Prototyping Station, *Proc. of the 3rd annual FPGAworld Conference*, Stockholm, Sweden, 2006.

[23] Hypertransport Consortium, *The Future of High Performance Computing: Direct Low Latency Peripheral-to-CPU Connections*, www.hypertransport.org, November 2005.