A Case for FPGA based Accelerated Communication

Holger Fröning, Mondrian Nüssle, Heiner Litz and Ulrich Brüning
Computer Architecture Group, Institute for Computer Engineering
University of Heidelberg
Mannheim, Germany
{holger.froening, mondrian.nuessle, heiner.litz, ulrich.bruening}@ziti.uni-heidelberg.de

***Abstract* — The use of Field Programmable Gate Arrays (FPGAs) in the area of High Performance Computing (HPC) to accelerate computations is well known. We present here a case where FPGAs can be used to speed up communication instead of computation. Current interconnects for HPC are in particular missing support for fine grain communication, which is increasingly found in various applications. In order to overcome this situation we developed a novel custom network. By using solely FPGAs it can easily be reconfigured to custom needs. The main drawback of FPGAs is their limited performance, which is about one to two orders of magnitude slower than commercial (specialized) solutions. However, an architecture optimized for small packet sizes results in a performance superior even to commercial high performance solutions. This excellent communication performance is verified by results from several popular benchmarks. In summary, we present a case where FPGAs can be used to accelerate communication and outperform commercial interconnection networks for HPC.**

Keywords- Computer Communications; High Performance Networking; Custom Interconnection Network

I. INTRODUCTION

High Performance Computing (HPC) today typically relies on two main components, which are the computational units and the *interconnection network (IN)*. Using these components large parallel systems with many nodes are built, in order to increase performance by parallelization. Due to the best price/performance ratio commodity server CPUs are used as computational units. However, the same does not apply to the interconnection network. The use of a standard commodity network like Gigabit Ethernet introduces severe limitations regarding performance and scalability. The main reasons for this are the requirements of an HPC system, which are low latency, high bandwidth, fault tolerance and scalability. As long as these issues are not addressed the costs for communication limit the overall performance of the system.

According to Amdahl's law [1] the sequential fraction of a program limits the maximum achievable speed-up by parallelization. The sequential fraction is directly related to the communication costs; or in other terms improving the communication costs increases the achievable speed-up. This can also be seen in the TOP500 list [2], which lists the theoretical peak performance as well as the sustained performance. With an optimized interconnection network the efficiency as the ratio between sustained and peak performance increases.

Several commercial interconnection networks for high performance computing exist. The important ones are Infiniband [3] [4], Cray's SeaStar [5] and Myrinet [6]. Typically, these INs are optimized for message passing as communication scheme, according to the *Message Passing Interface (MPI)* standard [6]. Adaption to custom requirements is very limited or completely impossible.

Message passing typically relies on the sending of large bulk data between nodes. Hence the INs above are optimized for high bandwidth and much less attention is paid to small packets. However, there are important applications which frequently use small packets in a fine grain communication pattern. Examples for such applications are the *RandomAccess* benchmark [12], computational fluid dynamics like *FLUENT* [8], the *Parallel Ocean Program* [9] or any *Partitioned Global Address Space (PGAS)* [10] application.

The main goal of this research project is to investigate new communication schemes and their impact on different applications, in particular the requirements of fine grain communication schemes. This is not possible with the limitations of the INs above. On the other hand, the performance should be comparable to the commercial solutions; otherwise the behavior of the overall system might be too different to draw conclusions from experiments.

In order to achieve these goals, in this work a custom high performance IN is developed which can be easily reconfigured to custom needs. Reconfigurability is achieved by using a *Field Programmable Gate Array (FPGA) [11]*. The challenge left is to achieve high performance with this technology, which is about one to two orders of magnitude slower than *Application Specific Integrated Circuits (ASICs*). In a summary, the main contributions of this work are:

- A novel custom high performance interconnection network
- Solely use of FPGA technology, facilitating custom communication units
- A complete MPI software environment
- Unmatched performance for an FPGA-based IN, exceeding Infiniband SDR for targeted packet sizes

The reminder of this work is structured as follows: The next section provides an overview about the related work. In section III, we present in detail the architecture of our

custom network. Results from different benchmarks are shown in the next section, while the last section concludes.

II. Related Work

High performance reconfigurable network interconnects are a highly relevant research topic. In [13] Schlansker et al. present an FPGA based Ethernet NIC which is plugged into an Intel Xeon processor socket enabling it to directly communicate with the processors over the front side bus. As the NIC resides in the cache coherent domain it is possible to transfer data using cache accesses which significantly lowers latency and increases bandwidth. While this approach would allow developing a whole new set of architectures and techniques, Schlansker focuses on the software side instead of exploring new hardware ideas.

Saldana et al. have developed High Performance Reconfigurable Computers using FPGAs as the main engine for computation and communication [14] [15]. By implementing a reduced MPI stack in hardware the FPGA can be directly controlled by software offloading many tasks that otherwise the host processor is responsible for. While implementing the software stack in hardware may provide significant performance gains it requires the modification of applications which rely on unsupported MPI functions. Due to the sheer amount of MPI functions a complete implementation of the MPI stack also seems unfeasible.

The Reconfigurable Computing Cluster project has the goal of developing a solely FPGA based petascale computing cluster [17]. The current implementation uses embedded PowerPC cores as the main computation engine combined with custom hardware for accelerating communication and for offloading certain MPI tasks. The disadvantage of this approach is its reliance on the PowerPC cores for running the full software stack which cannot compete with regular processors as a computation engine.

III. Architecture

The main goal of this work is to develop a custom interconnection network using FPGA technology, in order to support fine grain communication with high efficiency. In other terms, the overhead for small packets must be as small as possible. To allow optimizations throughout the IN, the complete IN hardware must be customizable. This requires the employment of FPGA technology for all elements in the network, respectively the network interface and the network switches.

We start with an analysis of the topology of the IN, which has significant impact on the optimizations for fine grain communication, as presented afterwards. The final architecture is integrated and implemented as an add-in card for computing nodes.

A. Topology

Typically, an IN is separated into the network interface which is residing as add-in card in the computing node and switching units, which are stand-alone devices not associated to any node. In such an indirect network multiple network interfaces are connected to a centralized switch. Most modern networks are indirect networks, for instance Ethernet, Infiniband [4] and Myrinet [6].

If a network interface provides multiple links towards the network side and in addition integrates the switch, direct networks are possible. No centralized switching resource is required any more; instead distributed switches (as part of the network interface) are used. Typical examples for such direct networks are meshes, tori and hypercubes. There are also some commercial implementations of direct networks, most noteworthy Cray's SeaStar [5] and IBM's BlueGene [16].

The advantage of direct networks for this work here is that only one unit has to be developed – integrating both the network interface and a decentralized switch. Another advantage is that in a centralized switch many ports are required in order to minimize the number of switches and the depth of the network. A 3D mesh or tori or a 6D hypercube requires only a link degree of 6 [18]. Including the internal connections 9 ports are necessary, which eases the design as the complexity of a single crossbar exhibits quadratic growth with the number of ports.

B. Optimizations for small packets

As one of the main goals is to support fine grain communication efficiently, the complete path from source to target node has to be optimized.

As a first step the coupling between network interface and the CPUs of a computing node is optimized. Unlike many other networks we are not using PCI-Express but the *HyperTransport (HT)* technology [19]. HT is typically found as a host interface between CPUs [20] and has excellent properties for low latency transmissions. The HTX standard allows plugging HT-based add-in card into an expansion slot – comparable to PCI-Express. Our measurements show latencies for PCI-Express devices of about 1000ns, and for HTX devices of about 180ns. Hence the HT interface is about 5 times faster in terms of latency compared to the PCI-Express interface.

Furthermore, the software overhead for packet processing is minimized by offloading most of the protocol from software to hardware. This results in an increased hardware complexity, but in particular no privileged software instance is required any more for packet processing. Communication processes can directly access the hardware using memory-mapped IO. This user level communication [21] avoids costly system calls. Another advantage of this is that efficient support for virtualization [22] [23] is possible, by allowing an arbitrary number of processes concurrent access to the network interface.

The use of direct networks increases the number of hops required to reach the destination. One of the major tasks is to minimize the switching latency, since overall latency increases linearly with the number of hops. Several techniques are used, which are in particular virtual output queuing [29], wormhole switching [24] [25], source path routing and credit-based flow control.

In order to minimize the overhead due to software layers two network properties are important. First, the network must maintain the ordering of packets, otherwise costly

operations in software layers are necessary to restore the original ordering. Second, the transmission must be reliable, i.e., as soon as a packet is sent out it is guaranteed that it will arrive correctly at its destination. Here, this is ensured by a link-level retransmission protocol, which automatically re-transmits faulty flits detected by a cyclic redundancy check (CRC) scheme. A reliable network allows the software layers to drop copies of the packet after sending; otherwise those copies have to be kept until a positive acknowledgement from the destination node is received.

C. *Top-Level Diagram*

The top level block diagram consists of several modules which can be separated in the host interface, several functional units in the network interface section and a switching unit with six links towards the network (Figure 1).

On the left side of the figure the host interface is shown, which is based on the HT Core [26]. It is connected to the functional units responsible for communication (VELO [27] and RMA [28]) using a custom On Chip Network (HTAX). VELO is optimized for small packets by minimizing communication overhead for send/receive operations. RMA supports the Put/Get model and is more suitable for large data transfers. An auxiliary unit for RMA is *Address Translation Unit (ATU)* [28], which provides a virtual-to-physical address translation mechanism for zero-copy, user level transactions.

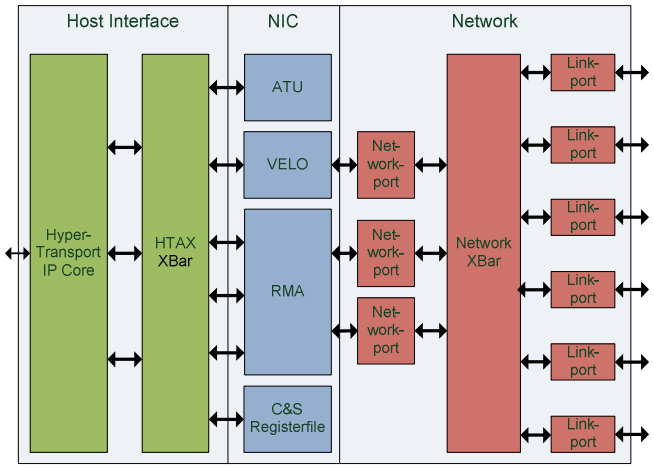

Figure 1. Top-Level Diagram of add-in card

The functional units are connected over the *Network Ports (NPs)* to the switch. In the NPs the conversion from internal protocol to network protocol takes place. The switch consists of a 9x9 bidirectional crossbar featuring round-robin arbiters to ensure fairness and in-order packet delivery. Three of the 9 ports of the crossbar are connected to the functional units; the other 6 are connected to the external links. The crossbar uses virtual output queuing [29], cut-through switching and credit-based flow control. All these techniques are used with latency minimization in mind.

The complete design from Figure 1 is implemented on a single FPGA located on an add-in card [30], which is shown in Figure 2. Most noteworthy in this figure is the FPGA in the center, the cage for the six link transceivers on the left and the HTX connector on the bottom.

The FPGA is running with a frequency of 200MHz for the HT-Core and 156MHz for the other core logic. The serial links are running at 6.24GHz and are using an 8b/10b coding. The internal data path has a width of 32bit, which at 156MHz matches exactly the serial link speed.

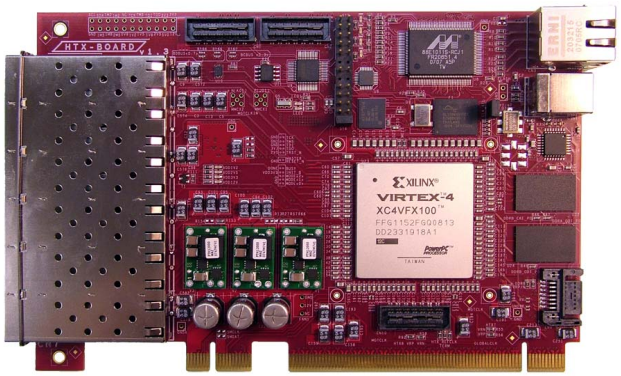

Figure 2. Add-in card implementing host interface, network interface and network switch

D. *Cases for Custom Configurations*

One of the goals of this work is a highly customizable design. However, the degree and ease of customization is difficult to characterize and quantify. Instead we want to describe a short example of a recent modification for custom needs in order to show the flexibility of our approach.

The goal of the modification is to increase the link speed for a small system, in which only three links are required. So, the link speed can be increased by bundling two links together, reducing the number of links from six to three. To achieve the necessary increase in speed inside the FPGA the internal data path width is doubled from 32 to 64 bit. The functional units as well as the interfaces to the switch (NPs and LPs) are modified to match the new width. The switch is a parametrizable unit which simply has to be re-generated. The on-chip network already has a data path width of 64bit, so no modifications are necessary for this module. Including design, simulation and verification the new design was realized in less than one week. After programming the FPGA with the new configuration the tests and benchmarks could be started immediately. Considering the impact of this modification – which is typically a major change in an application specific design – the required time for the modifications is very small.

Another example for a custom modification is the implementation of a functional unit for global shared memory access, which replaced the VELO and RMA unit for a specific series of experiments and measurements.

E. *Software Environment*

Open Fabrics Enterprise Distribution 1.4.1 [31] provided the low level Infiniband drivers and API libraries. For the custom network, custom low level drivers for Linux and API libraries have been developed. For both networks, OpenMPI 1.3 [32] was used as the MPI layer. Due to availability

reasons the Infiniband network uses OpenMPI version 1.3.2, while our custom network uses version 1.3.3. But according to the change log, the main differences are only bug-fixes.

The Infiniband network is used via the *Bit Transfer Layer (BTL)* of OpenMPI, while for the FPGA network a *Message Transfer Layer (MTL)* component was developed.

BTL components only provide data transport services; the necessary matching algorithms are implemented in the higher *point-to-point messaging layer (PML)*. An advantage of this approach is that PML can choose to use a different BTL for each peer process depending on the available connectivity. A classical example is to use shared memory for intra-node communication and the fastest available network for inter-node communication. This is exactly the case for the system used for this evaluation.

The MTL layer exports non-blocking point-to-point communication primitives and must implement MPI matching semantics inside the component. The MTL component for the FPGA component fully implements the necessary interfaces, but should still be considered experimental. Particularly, it has not been tested with large systems and is not yet optimally tuned for performance.

In the case of the custom FPGA network, all communication is routed over the FPGA, even if the peer process resides on the same node.

IV. Evaluation

The main goals of this work are a highly flexible custom design, which in addition offers a performance comparable to commercial (specialized) solutions. In this section we show the performance of our custom network by comparing it to an Infiniband network using several benchmarks.

Two nodes are used for benchmarking, each one consisting four quad-core Opterons (2.2GHz) and 16GB of Ram. Either one of these networks is used for benchmarking:

- Infiniband: Mellanox ConnectX IB SDR (MT25408), OpenMPI 1.3.2
- Custom Network: HT400 interface, 6.24Gbps link speed, Open MPI 1.3.3

Due to the network topology the Infiniband adapters are connected using a central switch, while our custom network is built as a direct network. This represents the topology of systems with more than two nodes. Considering only the theoretical performance, the IB network is far ahead of the custom network, providing a peak bandwidth of 1GB/s while our custom network only achieves a peak transfer rate of 624MB/s. However, the following experiments will show that in spite of this fact our custom network can outperform the IB network.

A. NetPIPE benchmark

In the first experiment we use the *NetPIPE* [33] benchmark to measure the message passing performance in terms of latency and bandwidth. The first experiment performs a Pingpong test. Here one node sends a packet to the other node, which responds with the same packet. The measured time on the first node divided by two is the half round trip latency.

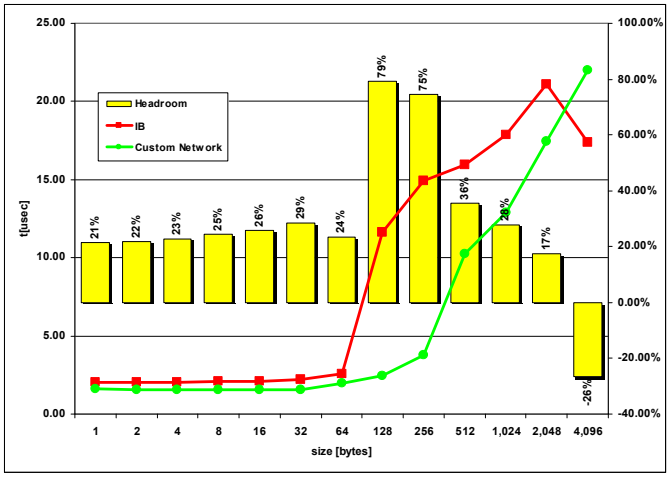

Figure 3. NetPIPE MPI Benchmark – Pingpong Latency

The first diagram (Figure 3) shows the measured half round trip latency for packet sizes of up to 4kB. This work targets an optimization for small packets, and the results clearly show the advantage of our solution compared to Infiniband. The headroom is the difference between our custom network and Infiniband as the percentage of Infiniband's latency. For instance, the latency for a packet size of 128B is 79% less when using our custom network, or in other terms 2.44µsec instead of 6.64µsec. Only for packets larger or equal than 4kB IB performs better due to its higher peak bandwidth.

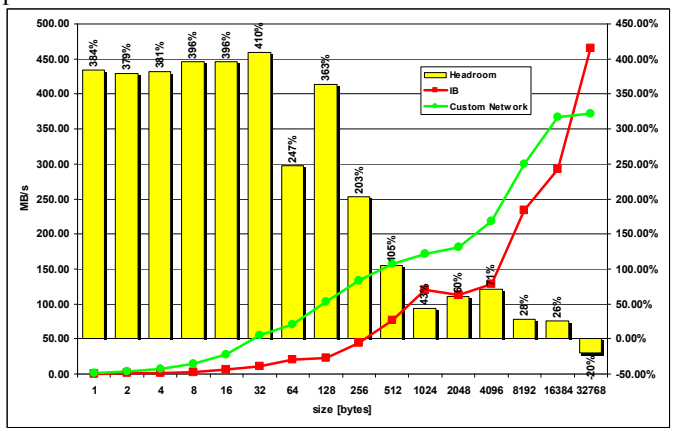

Figure 4. NetPIPE MPI Benchmark – Streaming Bandwidth

The diagram in Figure 4 shows the results from the Streaming test. Here, one node sends continuously packets to the other. No bidirectional communication takes place and performance data is measured as available bandwidth.

The measured bandwidth is up to 410% ahead of Infiniband, which is a factor of 5.1. The average headroom over the packet sizes shown is 217%. Only for packets larger or equal than 32kB the performance of IB is better. Considering an application which solely uses these packet sizes here, the communication part of the application achieves an average speed-up of 3.17x when using our custom network instead of the Infiniband network.

These measurements clearly show the advantage of our custom network in comparison with Infiniband, both in terms of latency and bandwidth for small packets.

B. *Intel MPI benchmark*

Another popular benchmark to characterize basic properties of a network is the Intel MPI benchmark suite [34]. We use this benchmark in addition to the NetPIPE Benchmark in order to verify the performance data.

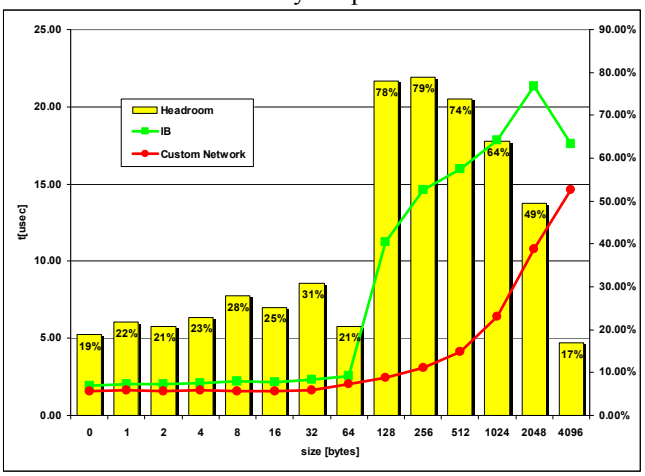

Figure 5. Intel MPI Benchmark – Pingpong Latency

The results are shown in Figure 5 and are very similar to the previous measurement, except for large packet sizes. For instance, at a size of 4kB Infiniband is no longer faster than our custom network. We did not investigate the internal behavior of NetPIPE for particular reasons for the degraded performance of large packets, because the results for sizes smaller or equal 256B are very similar to the previous ones.

C. *RandomAccess Benchmark*

With such excellent results from micro-benchmarks it can be estimated that the performance of real-world applications is also outstanding. One benchmark which models a real world application and makes use of small packet sizes is the *RandomAccess Benchmark [12],* which is part of the *High Performance Computing Challenge (HPCC)* [35] project. This benchmark measures the rate of integer random updates of memory.

The following diagram (Figure 6) shows the measured performance in *Billion (Giga) Updates per Second (GUPs)* for two selected table sizes. The bigger table size is the largest power of two that fits into the memory of the system.

It can be seen that our custom network achieves significantly more GUPS than the Infiniband network is able to deliver: for both table sizes the performance increase is about 2.8x. This in particular shows the advantage of our optimized custom network in a more complex benchmark.

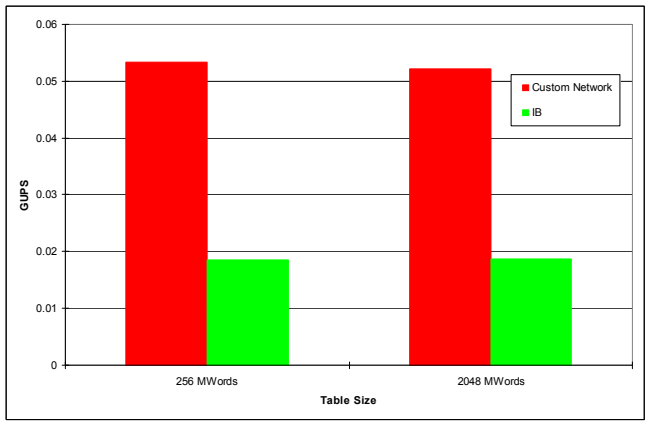

Figure 6. RandomAccess Benchmark – Giga Updates Per Second (GUPS)

V. Conclusion and Outlook

While it is well known that FPGAs can be used to speed up computations, we present here a case where FPGAs are used to speed up the communication of an HPC system. The FPGAs are used for a novel custom network with two main characteristics: First, the network is highly reconfigurable, i.e. it is easy to adapt either the network switch or interface to custom requirements. This in particular helps to develop new communication techniques by testing applications in a real world environment instead of a simulator. Second, the performance of our custom network is comparable to commercial available cluster interconnects. We are targeting fine grain communication schemes; hence we are only considering small packet sizes for this comparison.

Considering only the theoretical performance, the IB network provides a peak bandwidth of 1GB/s while our custom network only achieves 624MB/s. But for the targeted range of packet sizes the measurement results show that we are outperforming Infiniband significantly, both in terms of latency and bandwidth. We achieve latencies which are up to 79% less, and bandwidth is up to 5 times better than those of Infiniband. Only for packet sizes above 32kB Infiniband takes advantage of its higher link speed. As example for a more complex benchmark, the RandomAccess benchmark is executed about 2.8 times faster.

In the future we will further improve and fine tune our communication methods, in particular those based on the PGAS model. Our custom network is an optimal prototype station for all imaginable circumstances, and its outstanding performance allows performing tests and benchmarks without limitations.

Acknowledgements

We would like to express our thanks to all the people that contributed to this project, especially Dr. David Slogsnat, Sven Schenk, Niels Burkhardt, Alexander Giese and Benjamin Geib. We also thank Xilinx Inc. for their generous support.

REFERENCES

[1] Amdahl, G. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In *Proceedings of AFIPS Conference*, (30): 483–485, 1967.

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

[3] Infiniband Trade Association. *InfiniBand Architecture Specification Volume 1, Release 1.1*, 2002.

[4] Sur, S., Koop, M. J., Chai, L., and Panda, D. K. Performance Analysis and Evaluation of Mellanox ConnectX InfiniBand Architecture with Multi-Core Platforms. In *Proceedings of 15th Annual IEEE Symposium on High-Performance Interconnects (HOTI 2007)*, Aug. 2007.

[5] Brightwell, R., Pedretti, K., and Underwood, K. D. Initial Performance Evaluation of the Cray SeaStar Interconnect. In *Proceedings of the 13th Symposium on High Performance interconnects (HOTI)*, IEEE Computer Society, Washington, DC, 2005.

[6] Myrinet Myricom Inc. *Myrinet Express (MX): A High-Performance, Low-Level Message-Passing Interface for Myrinet*, Version. 1.2. 2006.

[7] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 1.3*, 2008.

[8] ANSYS, Inc. *ANSYS FLUENT Benchmark Suite*. http://www.fluent.com/software/fluent/fl6bench/index.htm, last visited Nov. 2009.

[9] Kerbyson, D. J. and Jones, P. W. A Performance Model of the Parallel Ocean Program. In *International Journal of High Performance Computing Applications,* 19(3), 261-276, 2005.

[10] Yelick, K., Bonachea, D., Chen, W., Colella, P., Datta, K., Duell, J., Graham, S. L., Hargrove, P., Hilfinger, P., Husbands, P., Iancu, C., Kamil, A., Nishtala, R., Su, J., Welcome, M., and Wen, T. Productivity and performance using partitioned global address space languages. In *Proceedings of the 2007 international Workshop on Parallel Symbolic Computation*, London, Ontario, Canada, July 2007.

[11] Xilinx Corporation. *Virtex-4 User Guide, Document ug070 v1.5*, www.xilinx.com, 2006.

[12] Aggarwal, V., Sabharwal, Y., Garg, R., and Heidelberger, P. HPCC RandomAccess benchmark for next generation supercomputers. In *Proceedings of the 2009 IEEE international Symposium on Parallel&Distributed Processing (IPDPS)*, IEEE Computer Society, Washington, DC, 2009.

[13] Schlansker, M., Chitlur, N., Oertli, E., Stillwell, P. M., Rankin, L., Bradford, D., Carter, R. J., Mudigonda, J., Binkert, N., and Jouppi, N. P. High-performance ethernet-based communications for future multi-core processors. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing - Volume 00,* ACM, New York, NY, 2007.

[14] Saldana, M., Patel, A., Madill, C., Nunes, D., Wang, D., Styles, H., Putnam, A., Wittig, R., and Chow, P. MPI as an abstraction for software-hardware interaction for HPRCs. In *Proceedings of the Second International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA)*, Austin, TX, 2008.

[15] Saldana, M., and Chow, P. TMD-MPI: An MPI Implementation for Multiple Processors Across Multiple FPGAs. *In Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, 2006.

[16] Adiga, N. et al. An overview of the BlueGene/L Supercomputer. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Baltimore, Maryland, 2002.

[17] Sass, R., Kritikos, W. V., Schmidt, A. G., Beeravolu, S., and Beeraka, P. Reconfigurable Computing Cluster (RCC) Project: Investigating the Feasibility of FPGA-Based Petascale Computing. In *Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE Computer Society, Washington, DC, 2007.

[18] Duato, J., Yalamanchili, S., and Ni, L. *Interconnection Networks: an Engineering Approach*, Morgan Kaufmann; 1st edition, 2002.

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

[20] Advanced Micro Devices (AMD). *AMD Opteron Product Data Sheet*, Publication #23932, 2004.

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

[22] Fröning, H. *Architectural Improvements of Interconnection Network Interfaces*. Ph.D thesis, Univeristy of Mannheim, Germany, 2007.

[23] Fröning, H., Litz, H., and Brüning, U. Efficient Virtualization of High-Performance Network Interfaces. In *Proceedings of the 2009 Eighth international Conference on Networks (ICN)*, IEEE Computer Society, Washington, DC, 2009.

[24] Chien, A. A. A Cost and Speed Model for k-ary n-Cube Wormhole Routers. In *IEEE Transactions on Parallel and Distributed Systems*, 9(2):150-162, 1998.

[25] Lysne, O. Deadlock Avoidance for Switches Based on Wormhole Networks, In *Proceeding of the International Conference on Parallel Processing (ICPP'99)*, 1999.

[26] Slogsnat D., Giese A., Nüssle M., and Brüning U. An Open-Source HyperTransport Core. In *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 1(3):1-21, 2008.

[27] Litz, H., Fröning, H., Nüssle, M., and Brüning, U. VELO: A Novel Communication Engine for Ultra-low Latency Message Transfers. In Proceedings of the *37th International Conference on Parallel Processing (ICPP-08)*, Portland, Oregon, USA, 2008.

[28] Nüssle M., Scherer, M., and Brüning, U. A resource optimized remote-memory-access architecture for low-latency communication. In Proceedings of the *38th International Conference on Parallel Processing (ICPP-09)*, Vienna, Austria, 2009.

[29] Mora, G., Flich, J., Duato, J., López, P., Baydal, E., and Lysne, O. Towards an efficient switch architecture for high-radix switches. In *Proceedings of the 2006 ACM/IEEE Symposium on Architecture For Networking and Communications Systems*, San Jose, California, USA, 2006.

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

[31] The OpenFabrics Alliance. http://www.openfabrics.org, last visited Nov. 2009.

[32] Gabriel, E., et al. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings of EuroPVM/MPI*, 2004.

[33] Turner, D., Oline, A., Chen, X., and Benjegerdes, T. Integrating New Capabilities into NetPIPE. In *Lecture Notes in Computer Science, Volume 2840/2003*, Springer Berlin / Heidelberg, 2003.

[34] Intel Corporation. *Intel MPI Benchmarks 3.2*. http://software.intel.com/en-us/articles/intel-mpi-benchmarks, last visited Nov. 2009.

[35] Luszczek, P., Bailey, D., Dongarra, J., Kepner, J., Lucas, R., Rabenseifner, R., and Takahashi, D. The HPC Challenge (HPCC) Benchmark Suite. In *IEEE SC06 Conference Tutorial*, Tampa, Florida, 2006.